Automatisation de la Méthode de Tour Into The Picture

Kévin Boulanger, Kadi Bouatouch

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Automatisation de la Méthode de Tour Into The Picture

Kevin Boulanger and Kadi Bouatouch

N° 5448

Janvier 2005

Thème COG

ISSN 0249-6399 ISRN INRIA/RR--5448--FR

Automatisation de la Méthode de Tour Into The Picture

Kevin Boulanger * and Kadi Bouatouch †

Thème COG — Systèmes cognitifs
Projets SIAMES

Rapport de recherche n° 5448 — Janvier 2005 — 64 pages

Résumé : La méthode de *Tour Into the Picture* (TIP) est une technique de rendu basé image. Son principe consiste à convertir une seule image en un ensemble de 5 polygones de coordonnées exprimées dans un repère 3D. A chacun de ces polygones est associée une texture représentant une partie de l'image redressée. Ensuite il est facile de naviguer dans la scène représentée par ces 5 polygones. Cette approche a deux inconvénients majeurs : le premier est l'obligation de régler un grand nombre de paramètres manuellement, le deuxième est l'ensemble de restrictions concernant la prise de vue originale (position et orientation). La méthode que nous proposons réduit au minimum les interventions de l'utilisateur et permet un maximum de liberté pour la prise de vue ; nous avons nommé cette méthode *Automatic Tour Into the Picture (ATIP)*. Nous proposons aussi des méthodes efficaces pour extraire des lignes et points de fuite nécessaires à l'étalonnage de caméra à partir d'une seule image.

Mots-clés : Tour into the picture, rendu basé image, étalonnage de caméra, points et lignes de fuite, rendu OpenGL

* Kevin.Boulanger@irisa.fr
† Kadi.Bouatouch@irisa.fr

Unité de recherche INRIA Rennes
IRISA, Campus universitaire de Beaulieu, 35042 Rennes Cedex (France)
Téléphone : +33 2 99 84 71 00 — Télécopie : +33 2 99 84 71 71

Automatic Tour Into The Picture

Abstract: *Tour Into the Picture* (TIP) is an image-based rendering method. It consists in converting one picture into a set of 5 polygons whose coordinates are expressed in a 3D coordinate system. With each polygon is associated a texture representing one rectified part of the picture. Then, it is easy to navigate through the scene represented by the 5 polygons. This approach, while giving convincing results, has two drawbacks: the first one is the necessity to tune manually a large number of parameters, the second is its constraint regarding the position and orientation of the camera which captured the picture. The method we propose reduce significantly the interventions of the user when extracting the set of 5 polygons and allows any position and orientation of the camera which captured the given picture. We have called this method: *Automatic Tour Into the Picture (ATIP)*. We also propose efficient methods for extracting vanishing lines and points useful for camera calibration from a single image.

Key-words: Tour Into The Picture, image-based rendering, camera calibration, vanishing points and lines, OpenGL rendering

Table des matières

1 Introduction

1.1 Le rendu basé image

La synthèse d'images comporte plusieurs domaines, et l'un d'eux s'appelle le rendu basé image (*Image Based Rendering (IBR)* en anglais). Ce mode de rendu utilise une ou plusieurs images et des techniques d'interpolation pour afficher une scène de bonne qualité visuelle (car souvent utilisé avec des photographies). A titre d'exemple, il existe une technologie nommée *Quicktime VR* qui permet, à partir d'une ou de plusieurs photographies prétraitées, d'observer une scène à 360 ° latéralement et verticalement si désiré. L'inconvénient majeur de cette méthode, même si les résultats sont convaincants, est l'impossibilité de se déplacer. Pour pouvoir effectuer une translation dans une scène, il faut la reconstruire avec des primitives (polygones, parallélépipèdes, ...) pour pouvoir se déplacer là où l'appareil photographique n'a pas été à l'origine. La reconstruction 3D est un domaine largement étudié, et qui a pour but de retrouver les primitives et les textures nécessaires au rendu de l'image finale.

Le résultat qui nous intéresse est destiné en particulier aux assistants de poche (PDA) pour un système de navigation par exemple. Les possibilités techniques de ces appareils sont limitées du fait de leur extrême compacité. Les techniques de reconstruction 3D classiques génèrent une quantité de données qui dépasse trop facilement les possibilités des PDA (une quantité trop élevée de polygônes provoque un ralentissement de l'affichage, un grand nombre de textures consomme une quantité exagérée de mémoire). Nous allons donc utiliser une méthode approximative qui permet de limiter au maximum les exigences techniques et qui permet d'obtenir un rendu satisfaisant sur un petit écran tel que celui d'un assistant de poche, par la même occasion sur un écran d'ordinateur.

1.2 Tour Into the Picture

D'après les travaux de [1, 2, 3, 4, 5], une nouvelle technique de rendu basé image a vu le jour et se nomme *Tour Into the Picture (TIP)*. Le principe est d'approcher le contenu d'une seule photographie par un parallélépipède unique composé de 5 quadrilatères. Les résultats obtenus sont plus convaincants que nous pourrions le penser. Par contre, cette approche a deux inconvénients majeurs : le premier est l'obligation de configurer un grand nombre de paramètres manuellement, le deuxième est l'ensemble de restrictions concernant la prise de vue originale (position et orientation). La méthode que nous proposons réduit au minimum les interventions de l'utilisateur et permet un maximum de liberté pour la prise de vue ; nous avons nommé cette méthode *Automatic Tour Into the Picture (ATIP)*. Un résultat est donné sur la figure 2 qui est une vue reconstruite à partir de la photographie de la figure 1. D'autres résultats sont donnés sur les figures 37 et 38.

La méthode du *Tour Into the Picture* est développée dans [1, 2], le principe est simple mais permet d'obtenir des résultats encourageants. Le premier traitement consiste à séparer

FIG. 1 – Photographie originale

FIG. 2 – Rendu de la scène depuis un autre point de vue

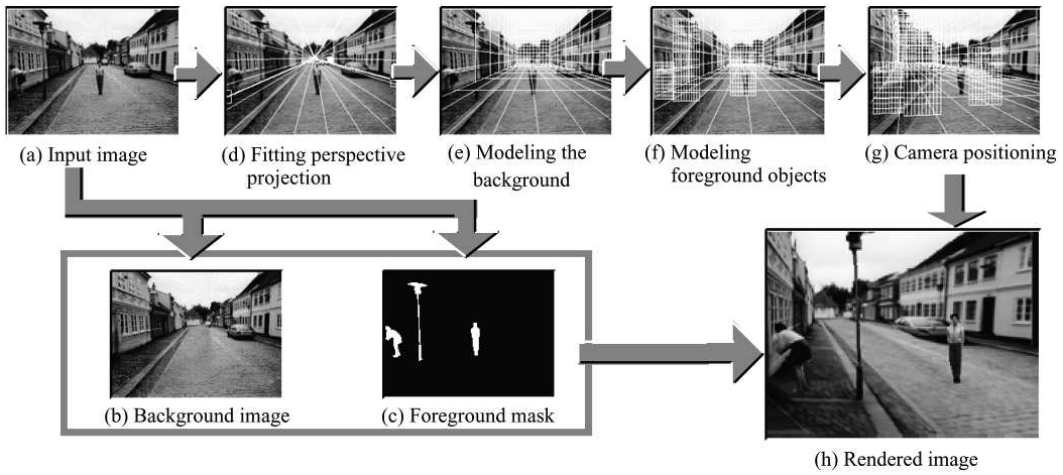

FIG. 3 – Etapes du *Tour Into the Picture* original, images provenant de [1]

les objets en avant-plan de l'arrière-plan. Pour cela, un masque est utilisé pour sélectionner les objets en avant-plan (figure 3*(c)*). Une fois ces objets retirés de l'image originale (figure 3*(a)*), il reste l'arrière-plan et des espaces vides à compléter. Pour cela, un logiciel de traitement d'images est adapté en utilisant le pinceau de clonage (figure 3*(b)*). Trois images sont obtenues (image originale, masque et arrière-plan complété) qui servent pour la suite des traitements. A partir de ce moment, il faut positionner manuellement le point de fuite dans l'image (figure 3*(d)*) à l'aide d'un *spidery mesh* pour déterminer la projection perspective associée. Ensuite une grille peut être déterminée (figure 3*(e)*) qui permet de déduire des quadrilatères (5 au maximum). Des textures doivent être calculées à partir de l'image originale pour être plaquées sur ces quadrilatères lors du rendu. Les objets en avant-plan sont rendus grâce à des *billboards* texturés (c'est-à-dire des quadrilatères toujours alignés par rapport au plan de l'écran) (figure 3*(f)*). Enfin, la caméra est repositionnée de manière à observer la scène d'un autre point de vue (changement de position et d'orientation) (figures 3*(g)* et 3*(h)*).

Malgré les déformations provoquées par les approximations (car la scène n'est quasiment jamais un parallélépipède parfait), le résultat est convaincant. Cette méthode possède malheureusement plusieurs contraintes importantes. La première est le temps passé pour le pré-traitement avec un logiciel de traitement d'images. Cette étape n'est pas obligatoire car le rendu de l'arrière-plan uniquement pourrait suffire. Le plus gros défaut est certainement la présence de contraintes concernant la prise de vue : les horizontales et les verticales doivent être bien parallèles aux bords de l'image, ce qui implique que la photographie doit être prise en étant parfaitement parallèle au sol, aux murs. Ceci est extrêmement difficile avec un appareil photographique tenu dans la main (même en ayant l'impression de se positionner correctement, nous retrouvons toujours une rotation de quelques degrés (généralement entre

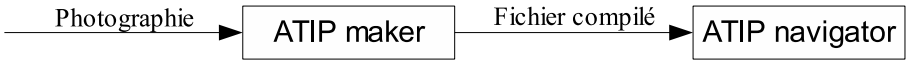

FIG. 4 – Les deux logiciels et leurs entrées-sorties

1 ° et 5 °)). Il faut donc un support pour fixer l'appareil photo, pas toujours pratique pour prendre des photographies dans la rue par exemple. Ceci implique que la photographie ne doit comporter qu'un seul point de fuite à l'intérieur ou à l'extérieur de l'image.

Malgré cela, nous pouvons quand même trouver un avantage à l'intervention d'un utilisateur pour cette méthode : cela permet de traiter n'importe quel type d'image (photographie, dessin, peinture, etc. ...), cela permet aussi le traitement des objets en avant-plan.

2 Automatisation du Tour Into the Picture

2.1 Architecture

Notre objectif est d'automatiser le maximum d'étapes de l'algorithme *Tour Into the Picture* original, de manière à :
- soulager l'utilisateur des nombreuses manipulations à effectuer,
- réduire le temps de traitement,
- permettre l'utilisation de photographies effectuées sous n'importe quel angle.

Automatic Tour Into the Picture est composé de deux logiciels (résumés sur la figure 4) :
- *ATIP maker* : permet d'effectuer un ensemble de traitements sur une image de manière à créer un format de fichier compilé contenant toutes les données nécessaires pour naviguer dans la scène,
- *ATIP navigator* : utilise les fichiers compilés par *ATIP maker* pour naviguer dans la scène. Ce logiciel peut être décliné en plusieurs versions, pour plusieurs plateformes matérielles (PC, PDA, ...).

La plus grande partie du travail a consisté à concevoir et implémenter *ATIP maker*, car il contient la plus grande partie théorique, aussi bien en traitement qu'en synthèse d'image. Après l'étude bibliographique, 3 manières de concevoir le logiciel ont été retenues ; elles sont résumées sur la figure 5 :
- la première méthode (chemin *(abcghi)*) est constituée d'un ensemble de traitements simples consécutifs, mais qui demandent d'effectuer un grand nombre de réglages, en particulier au niveau des seuils,
- la deuxième méthode (chemin *(adghi)*) cherche les points de fuite de l'image par une méthode probabiliste en utilisant une fonction dépendant du voisinage. Cette méthode,

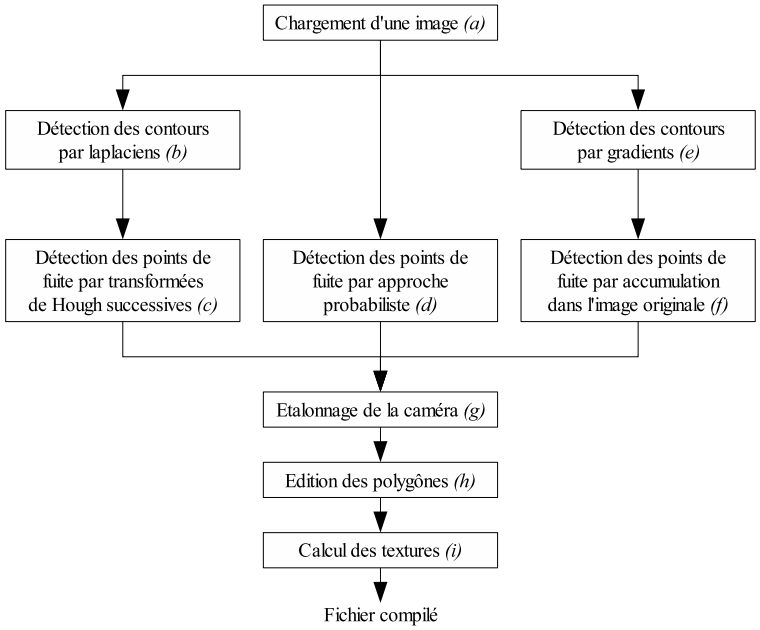

FIG. 5 – Architecture d'*ATIP maker*

expliquée dans [6], prometteuse au niveau de la qualité des résultats, demande une grande quantité de calculs, conséquence que nous voudrions éviter,

– la troisième méthode (chemin *(aefghi)*) est simple à implémenter, rapide, mais est malheureusement très imprécise, et ne gère pas tous les cas de positionnement de la caméra originale. Elle est expliquée dans [7].

La solution que nous avons retenue est la première. Elle est assez longue à implémenter par rapport aux deux autres car elle contient un grand nombre de traitements mais ils sont décomposables en un ensemble de petits traitements en général assez simples. Chacun de ces traitements peut être optimisé individuellement dans le but d'obtenir des performances globales correctes.

2.2 Analyse de l'image originale

Le premier grand bloc fonctionnel d'*ATIP maker* est l'analyse de l'image originale de manière à en extraire les points de fuite. La solution que nous avons retenue, expliqué précédemment est constituée d'une ensemble de traitements décrits ci-après.

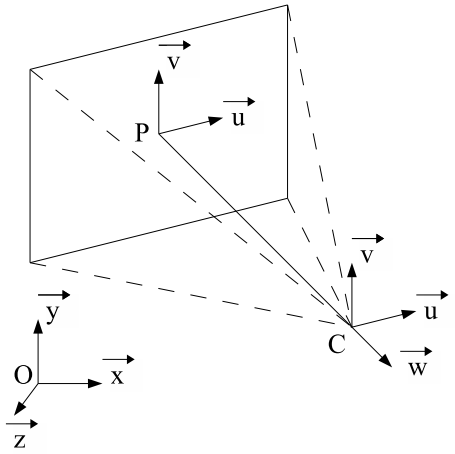

FIG. 6 – Description d'une caméra

2.2.1 Rappels sur les notions de perspective

Avant d'entrer dans les détails de la méthode, voici quelques indications concernant les notions de perspective et la manière dont nous nous en servons par la suite.

La première notion importante est celle de *caméra*. Elle représente l'observateur et comporte quelques paramètres :

- les paramètres *extrinsèques* : ce sont les plus évidents. Il s'agit de la position, de l'orientation de la caméra,
- les paramètres *intrinsèques* : ils caractérisent la caméra. Il s'agit principalement de la focale, nous y trouvons aussi le ratio des pixels et les coordonnées du point principal. Nous avons retenu un ratio de 1 (pixels carrés), le point principal est fixé au centre de l'image.

Les approximations retenues ne permettent pas d'étalonner la caméra aussi précisément que dans les algorithmes classiques de vision par ordinateur. La raison principale est la quasi-impossibilité de photographier une mire pour étalonner l'appareil photographique dans les situations les plus courantes, dans la rue par exemple. Si deux photographies sont prises avec l'appareil dans les mains, une avec la mire et une sans, les paramètres intrinsèques ne sont pas modifiés. A l'inverse, les paramètres extrinsèques sont modifiés à cause de l'instabilité de notre position. Le résultat final du *Tour Into the Picture* est déjà une approximation grossière de la scène, l'étalonnage très précis de la caméra n'est donc pas justifié.

Sur la figure 6, le repère global est $(O, \vec{x}, \vec{y}, \vec{z})$, le repère de la caméra est $(C, \vec{u}, \vec{v}, \vec{w})$. Le vecteur $\overrightarrow{OC}$ représente la translation de la caméra et la matrice $(\vec{u}\ \ \vec{v}\ \ \vec{w})$ la rotation de la

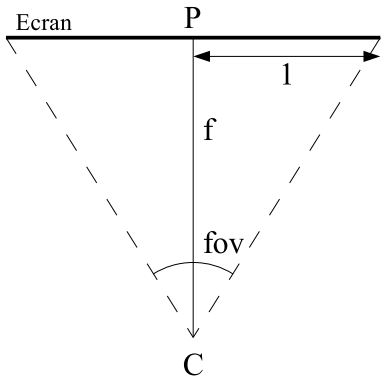

FIG. 7 – Caméra - Vue de dessus

caméra par rapport au repère global. Le point P est le *point principal* de la caméra, il est fixé au centre de l'écran comme expliqué précédemment. La distance $f = \|\overrightarrow{CP}\|$ est la *focale* de la caméra.

Un paramètre supplémentaire peut être déduit de la focale, il s'agit du champ de vision (*FOV* en anglais pour *Field Of View*), il s'agit de l'angle d'ouverture de la caméra. Sur la figure 7, nous voyons la caméra en vue de dessus, et le champ de vision est $fov = 2\arctan(\frac{1}{f})$.

Définissons ce que nous appellons projection perspective. Quand une scène à 3 dimensions est affichée sur un écran, il faut réaliser une projection sur un espace à 2 dimensions. Sur la figure 8, nous pouvons observer la projection du point de coordonnées (x, y, z) sur l'écran. Les coordonnées du point projeté sont, dans le repère de la caméra $(C, \vec{u}, \vec{v}, \vec{w})$:

$$\begin{pmatrix} x_e \\ y_e \\ z_e \end{pmatrix} = \begin{pmatrix} x\frac{-f}{z} \\ y\frac{-f}{z} \\ -f \end{pmatrix}$$

Quand un ensemble de droites parallèles infinies sont projetées de cette manière sur l'écran, ces droites convergent en un point nommé *point de fuite fini*. Si, dans une image donnée, ce point de fuite peut être trouvé, il donne des indications sur les directions des droites dans l'espace 3D qui permettent de déterminer les vecteurs du repère de la caméra. C'est cette propriété que nous utilisons pour étalonner la caméra. Si le vecteur directeur des droites parallèles se trouve dans le plan de l'écran alors nous parlons de *point de fuite infini*, le seul élément détectable sur l'image projetée est un ensemble de droites parallèles et non un point. Sachant qu'un point de fuite, fini ou infini, donne une indication concernant une direction dans l'espace, la connaissance de trois points de fuite permet donc de déterminer complètement les directions des vecteurs du repère de la caméra, à condition que les trois

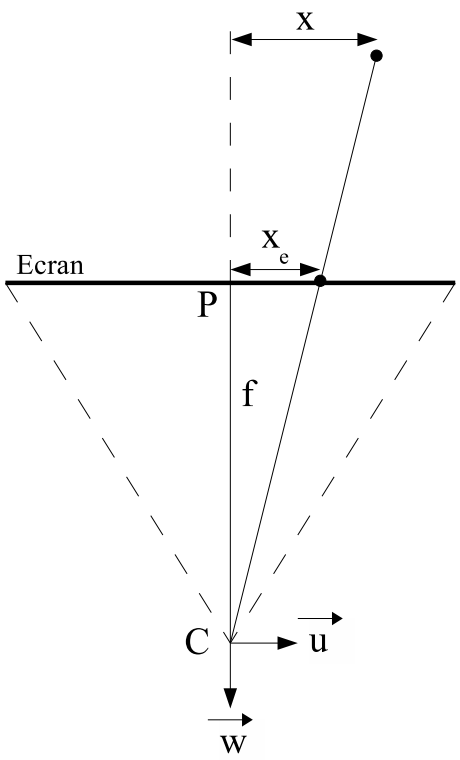

FIG. 8 – Projection perspective d'un point sur l'écran

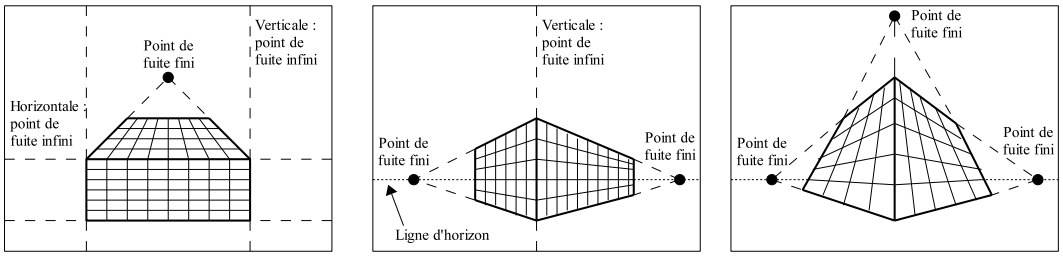

FIG. 9 – Les trois combinaisons de points de fuite finis et infinis

directions soient orthogonales. Plusieurs combinaisons sont possibles, montrées sur la figure 9 :

- *1 point de fuite fini, deux points de fuite infinis* : cette situation est obtenue si deux directions de l'espace 3D se trouvent dans le plan de l'écran, par exemple si une image est photographiée « bien droit ». L'exemple typique est la photographie d'une rue dans la longueur, où les horizontales sont horizontales dans le plan de l'écran, de même pour les verticales. Une rotation autour de la normale au plan de l'écran ne change pas la configuration,
- *2 points de fuite finis, un point de fuite infini* : cette situation est obtenue si une seule direction de l'espace 3D se trouve dans le plan de l'écran, par exemple si l'image a été photographiée en se tenant debout, bien vertical mais sans alignement avec les lignes horizontales du sol,
- *3 points de fuite finis* : cette situation est obtenue si aucune direction de l'espace 3D ne se trouve dans le plan de l'écran, c'est-à-dire si la photographie a été prise sans aucun alignement.

Chacun de ces cas doit être traité séparément lors de l'étalonnage de la caméra car ils possèdent chacun des propriétés différentes. Sur la figure 9, chaque image est considérée avec la verticale parallèle aux bords gauche et droit de l'écran ; une rotation autour de la normale au plan de l'écran ne change rien à la configuration des points de fuite ; cette propriété sera exploitée lors de l'étalonnage de la caméra. Les lignes passant par les points de fuite finis sont appelées *lignes de fuite*, tout comme les lignes parallèles aux directions des points de fuite infinis.

2.2.2 Chargement de l'image originale

La première étape du traitement concerne le chargement de l'image originale. Cette image peut-être une illustration, une photographie, le critère principal étant la présence de lignes de fuite permettant l'étalonnage de la caméra. Plus la résolution de l'image est élevée, plus le résultat final est précis mais plus le temps de calcul augmente. Une image en 1024×768 donne de bons résultats, mais les textures finales sont un peu floues. Une image en 2592×1944 (5 mégapixels) permet d'obtenir des textures très nettes. Pour manipuler des images de cette taille, il est préférable qu'elles soient compressées avec un codage *JPEG* par exemple. Le taux de compression ne doit pas être trop élevé de manière à ce que les artefacts de compression nuisent le moins possible à la détection des lignes de fuite.

Le premier traitement après le chargement est la conversion en niveaux de gris en utilisant la formule suivante :

$$G = 0.299R + 0.587V + 0.114B$$

(R, V, B) étant la couleur $(Rouge, Vert, Bleu)$ originale et G le niveau de gris final. Cela permet d'obtenir une image de luminance qui est utilisée pour la détection des contours. Si l'image originale est déjà en niveaux de gris, la conversion n'est pas nécessaire.

2.2.3 Détection des contours

Pour détecter les points de fuite de l'image, nous utilisons la présence de lignes de fuite dans l'image, pour cela des droites ou des segments doivent être extraits de l'image. Il existe une grande quantité d'algorithmes permettant d'extraire ces primitives, les principaux sont basés sur du filtrage 2D. Les *contours* des objets sont considérés comme les endroits où de fortes variations de luminance ont lieu. Nous utilisons au choix deux catégories d'algorithmes donnant des résultats différents, l'une basée sur les laplaciens, l'autre sur les gradients.

Méthode du laplacien :

Elle consiste à utiliser un filtre qui dérive deux fois les composantes de l'image (figure 10*(a)*). Pour cela un filtre de convolution 2D est utilisé ; plusieurs noyaux sont possibles et nous utilisons le noyau suivant car il donne de bons résultats :

$$\frac{1}{8}\begin{bmatrix} -2 & 1 & -2 \\ 1 & 4 & 1 \\ -2 & 1 & -2 \end{bmatrix}$$

Ce filtre n'est pas séparable directement donc demande une grande quantité de calculs. Nous pouvons remarquer que ce filtre est la somme de deux filtres 2D séparables :

$$\frac{1}{8}\begin{bmatrix} -2 & 1 & -2 \\ 1 & 4 & 1 \\ -2 & 1 & -2 \end{bmatrix} = \frac{1}{8}\begin{bmatrix} -1 & 2 & -1 \\ -1 & 2 & -1 \\ -1 & 2 & -1 \end{bmatrix} + \frac{1}{8}\begin{bmatrix} -1 & -1 & -1 \\ 2 & 2 & 2 \\ -1 & -1 & -1 \end{bmatrix}$$

Chacun des deux termes de la somme représente le filtrage suivant une direction (laplaciens horizontaux et verticaux). Ces deux filtres sont séparables et utilisent les noyaux $[-1\ 2\ -1]$ et $1/8\ [1\ 1\ 1]$. Les temps de calcul sont donc diminués.

Une fois les laplaciens calculés, nous obtenons une image où les niveaux représentent les courbures (nombres positifs et négatifs). Un passage par zéro représente un changement de cette courbure, c'est ce que nous voulons détecter. Cette étape (figure 10*(b)*) renvoie une image binaire (figure 11*(b)*) contenant des *pixels blancs* représentant les passages par zéro. L'étape suivante consiste à mémoriser les pixels les plus importants, en l'occurrence ceux pour lesquels la norme des gradients de l'image originale est supérieure à un seuil. Ces forts gradients représentent les changements importants de luminance dans l'image originale en niveaux de gris, et donc les contours.

Selon le type de seuillage effectué (figures 10*(d)*, 10*(e)*, 10*(g)*), les gradients sont soit calculés sur les zéros du laplacien (figure 10*(c)*), soit sur l'image complète (figure 10*(f)*).

Pour calculer les gradients de l'image, nous utilisons un filtre de convolution 2D isotrope, c'est-à-dire qui ne favorise aucune direction particulière ; la direction des gradients calculés

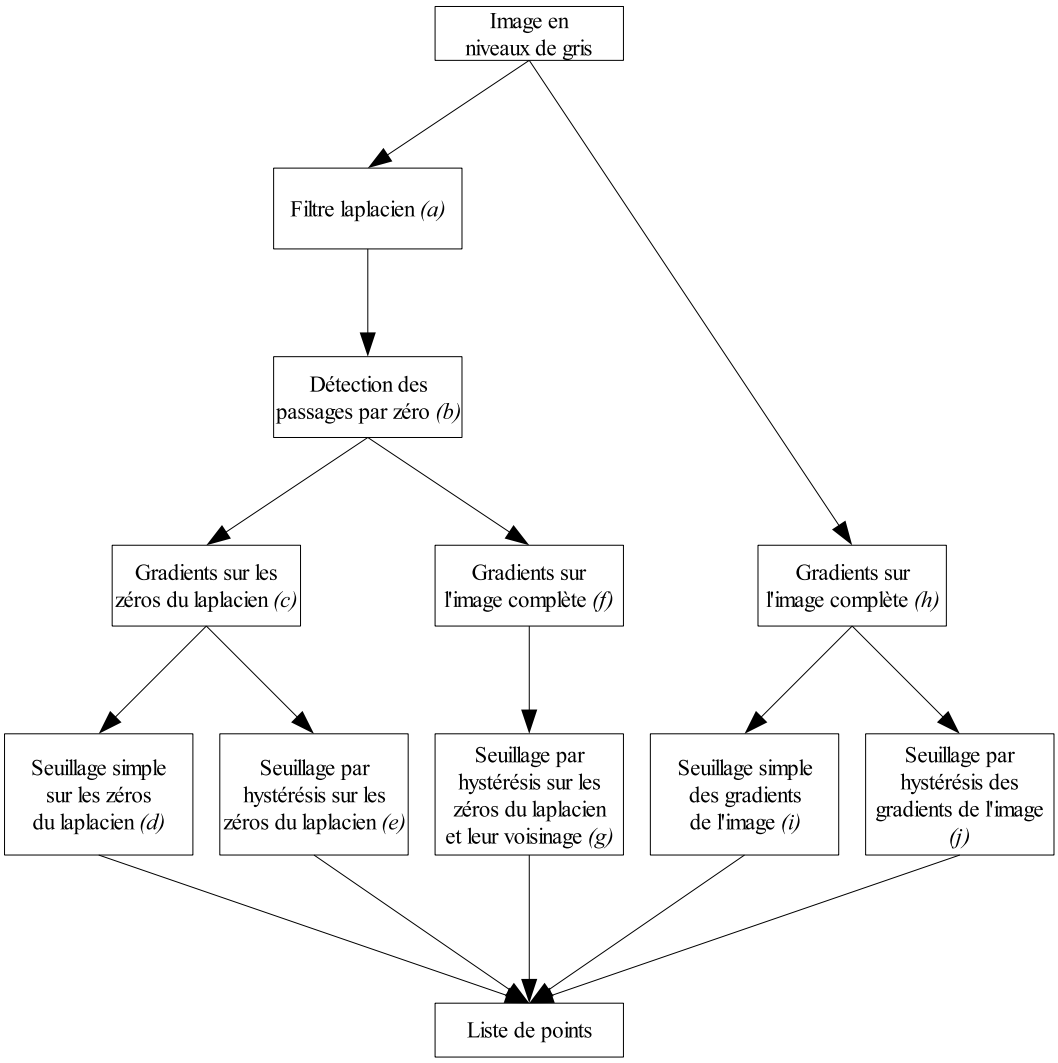

FIG. 10 – Procédures possibles pour la détection des contours

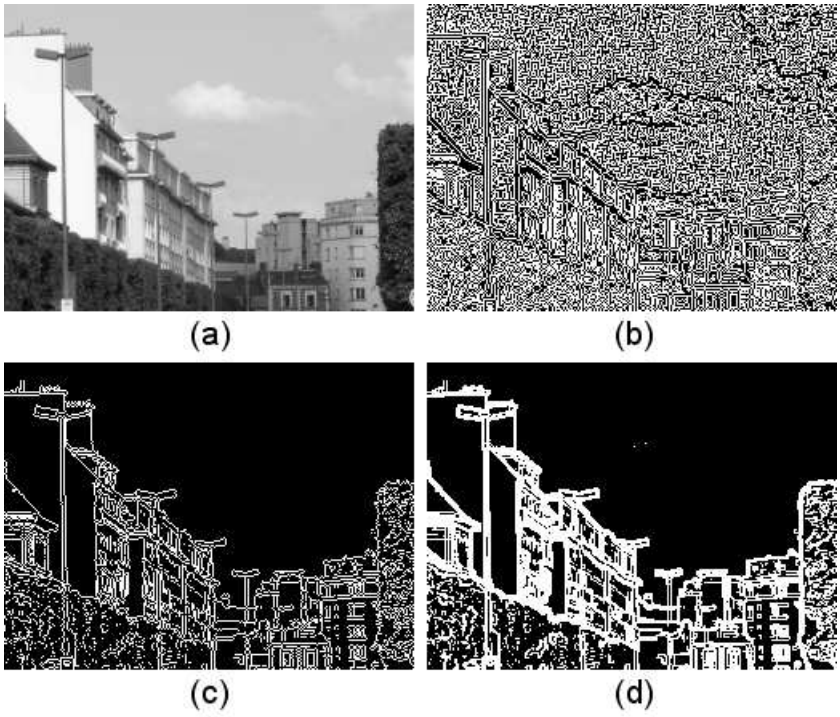

FIG. 11 – Détection des contours (*(a)* image en niveaux de gris, *(b)* passages par zéro des laplaciens, *(c)* seuillage simple des zéros des laplaciens, *(d)* seuillage simple de la norme des gradients)

est un paramètre important pour déterminer les directions des lignes de fuite. Le filtre de Frei-Chen appartient à cette catégorie, son noyau pour la détection de la composante horizontale est le suivant :

$$\frac{1}{2+\sqrt{2}} \begin{bmatrix} 1 & 0 & -1 \\ \sqrt{2} & 0 & -\sqrt{2} \\ 1 & 0 & -1 \end{bmatrix}$$

Pour la composante verticale, nous utilisons :

$$\frac{1}{2+\sqrt{2}} \begin{bmatrix} -1 & -\sqrt{2} & -1 \\ 0 & 0 & 0 \\ 1 & \sqrt{2} & 1 \end{bmatrix}$$

Ce filtre n'est pas séparable, mais ce n'est pas vraiment utile dans notre cas par la présence de nombreux 0 et 1. Les 0 n'interviennent pas dans les calculs, les 1 représentent de simples additions, et les -1 de simples soustractions. Le coefficient de normalisation peut être mis sous forme de constante dans le code source pour ne pas avoir à le recalculer inutilement. Par contre, la présence des racines carrées nous force à convertir l'image en niveaux de gris en nombres flottants (procédure coûteuse en temps de calcul) et à utiliser des opérations sur les nombres flottants pour le calcul du filtre (coûteuses aussi). Le temps de calcul n'est finalement pas trop élevé par rapport à la détection des lignes de fuite par exemple, c'est pour cela que nous retenons quand même cette méthode.

Nous voulons effectuer un seuillage sur la norme des gradients (figures 10*(d)*, 10*(e)*, 10*(g)*). La norme que nous retenons est la norme L_2 car elle est calculée de la même manière pour toutes les directions, ceci est important pour la prise en compte des lignes de fuite. L'utilisation du filtre isotrope de Frei-Chen est justifiée car le calcul de cette norme ne doit pas être gêné par un filtre de convolution anisotrope.

Le seuillage simple (figure 10*(d)*) consiste à ne conserver que les zéros du laplacien pour lesquels la norme du gradient est supérieure à un seuil. C'est la méthode la plus rapide et donne de bons résultats en général.

Le seuillage par hystérésis (figure 10*(e)*) utilise deux seuils, le bas et le haut, et fonctionne comme suit : si la norme du gradient est supérieure au seuil haut, le pixel est retenu ; si la norme est inférieure au seuil bas, le pixel est rejeté ; sinon, si au moins un voisin du pixel a la norme de son gradient supérieure au seuil haut, le pixel est retenu. Les voisins considérés ne sont que les zéros du laplacien dans le 8-voisinage du pixel courant. Cette méthode rejette moins de pixels que le seuillage simple pour les gradients forts, par contre rejette plus de gradients faibles, écartant donc plus le bruit.

Le seuillage par hystérésis peut aussi se servir du 8-voisinage complet (figure 10*(g)*) (comme décrit dans [8]) mais nécessite le calcul des gradients pour toute l'image (plus optimisé que le calcul des gradients des 8 voisins pour chaque zéro du laplacien, car ne

calcule le gradient qu'une seule fois pour chaque pixel). Les résultats obtenus par cette méthode sont similaires au seuillage par hystérésis avec le voisinage partiel et n'est donc pas vraiment justifiée car consomme plus de temps de calcul.

Méthode des gradients :

Elle est en fait un sous-ensemble de la méthode du laplacien. Dans un premier temps, les gradients sont calculés sur l'image complète (figure 10*(h)*) par le filtre de Frei-Chen abordé précédemment, puis la norme L_2 est calculée pour chaque pixel. La méthode du seuillage simple consiste à ne retenir que les pixels dont la norme du gradient est supérieure à un seuil (figure 11*(d)*). La méthode du seuillage par hystérésis fonctionne comme décrite précédemment et le 8-voisinage des pixels est utilisé.

Methode retenue :

Toutes les méthodes décrites ont leurs avantages et leurs inconvénients et sont donc implémentées dans *ATIP maker*. Une méthode est utilisée par défaut : celle du laplacien avec le calcul des gradients sur les zéros du laplacien et seuillage simple (chemin *(abcd)* sur la figure 10 et résultats sur la figure 11).

La méthode du laplacien renvoie des contours de 1 pixel de largeur (figure 11*(c)*). Les directions des contours sont donc plus précises, contrairement à la méthode des gradients pour laquelle les contours comportent plusieurs pixels en largeur (figure 11*(d)*). De plus, le nombre de points dans la liste finale est plus faible, accélérant de manière significative la détection des lignes de fuite.

Le seuillage simple a été retenu car ce qui est considéré comme *bruit* dans le seuillage par hystérésis et qui est rejeté peut en fait être souvent un ensemble de petits segments se trouvant sur les lignes de fuite. Ces petits segments améliorent en général la détection de ces lignes de fuite.

2.2.4 Détection des lignes de fuite

Comme expliqué dans la section 2.1 et sur la figure 5, nous avons décidé de détecter les points de fuite par transformées de Hough successives (détection des lignes de fuite par transformée de Hough polaire, puis détection des points de fuite par projection sur l'hémisphère gaussienne). Le but de la première transformée est de trouver les lignes de fuite, la seconde les points de fuite. Nous nous intéressons pour le moment à la première.

Il existe plusieurs types de transformées de Hough selon le type de primitive à manipuler ; dans notre cas, il s'agit des droites (pour détecter les lignes de fuite). La transformée

de Hough permet de passer d'un espace tel qu'une image à un espace de paramètres. Par exemple, si une droite dans l'espace a pour équation $y = ax + b$ alors la transformée donne le point (a, b).

Il existe deux catégories de transformée de Hough :
- la transformée *m vers 1* qui consiste à transformer un ensemble de droites en un nuage de points (méthode abordée dans [9])
- la transformée *1 vers m* qui transforme un ensemble de points en courbes (dans notre cas).

Nous utilisons la deuxième méthode car la première est extrêmement consommatrice de temps de calcul : elle doit exécuter un calcul pour toutes les paires de points des contours détectés dans l'image. La deuxième méthode demande des calculs plus intensifs pour chaque point mais ne le fait qu'une fois et à temps constant. Avec quelques optimisations d'implémentation, nous obtenons une vitesse d'exécution satisfaisante.

Commençons par décrire la primitive que nous allons traiter : la droite. La représentation cartésienne (c'est-à-dire le couple (a, b) tiré de l'équation $y = ax + b$) ne permet de pas de représenter avec la même précision les différentes orientations. Il n'est d'ailleurs pas possible de représenter correctement une droite parallèle à l'axe y ; c'est pour cela que nous optons pour une représentation polaire. Une droite est paramétrée par deux valeurs : ρ et θ, comme décrit sur la figure 12*(a)*. Cette fois, toutes les orientations sont possibles avec une précision constante, et la distance par rapport à l'origine permet de placer la droite dans l'espace. Nous aurions pu utiliser des segments au lieu des droites, mais leur détection est plus compliquée ; leur usage est plutôt réservé à la reconstruction 3D, il n'est pas justifié pour la détection de lignes de fuite. Pour détecter des segments, nous pouvons dans un premier temps détecter des droites, puis déterminer quels sont les pixels de ces droites appartenant aux contours ; cette manipulation est un peu plus longue en temps d'exécution, mais le problème provient surtout de la possible présence de nombreux segments parasites, à cause des feuilles d'un arbre par exemple. La faible longueur de certains segments peut apporter aussi beaucoup d'imprécision sur leur orientation.

L'espace de la transformée de Hough que nous allons utiliser est représenté sur la figure 12*(b)*. Il est discrétisé de manière à être un tableau à 2 dimensions avec un nombre important de *cellules*. Sur l'axe des abscisses, nous trouvons les différentes orientations possibles d'une droite (allant de 0 à 2π radians). Sur l'axe des ordonnées, la distance d'une droite à l'origine varie de $-\rho_{max}$ à $+\rho_{max}$. Cette distance maximum peut être calculée ainsi ($largeur$ et $hauteur$ étant les dimensions de l'image originale en pixels) :

$$\rho_{max} = \left\lceil \sqrt{largeur^2 + hauteur^2} \right\rceil$$

Au départ, toutes les cases sont initialisées à 0 (valeur entière). A chaque point de l'image de contours sera associée une courbe tracée dans cet espace ; chaque case recouverte par la

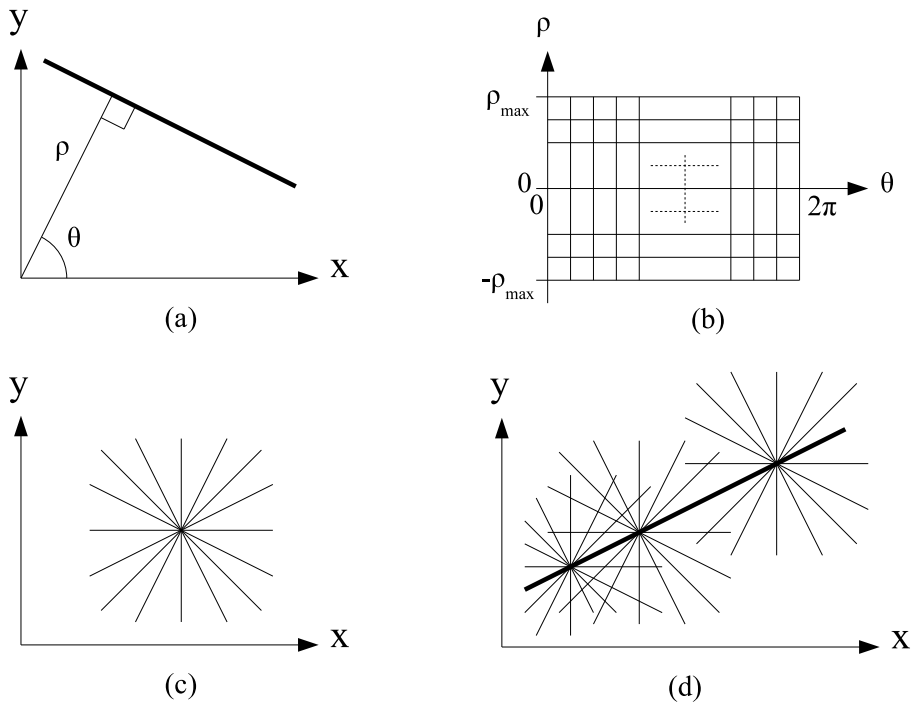

FIG. 12 – Principe de la transformée de Hough utilisée *((a)* paramètres polaires d'une droite, *(b)* espace de Hough utilisé, *(c)* définition d'un point comme l'intersection d'une infinité de droites, *(d)* principe de détection des points alignés)

courbe est incrémentée de 1. Ce principe de comptage s'appelle *accumulation.* Les dimensions de la grille d'accumulation ne doivent pas être trop petites (manque de précision), ni trop grandes (accumulation insuffisante dans les cellules). Elles peuvent être fixées par l'utilisateur ou spécifiées automatiquement. Dans ce dernier cas, nous choisissons de fixer le nombre d'orientations possibles des droites à la largeur de l'image en pixels.

Un point de l'image de contours est considérée comme l'intersection d'une infinité de droites (figure 12*(c)*). Sachant que chaque droite forme un point dans l'espace de Hough, une infinité de droites forme une courbe d'équation :

$$\rho = x\cos(\theta) + y\sin(\theta)$$

(x, y) étant les coordonnées en pixels du point à transformer.

Pour pouvoir détecter les droites présentes dans l'image, il faut détecter les principaux alignements de points. Lorsque des points (intersections d'une infinité de droites) sont alignés, il existe une droite commune (figure 12*(d)*) pour laquelle la case correspondante dans la transformée de Hough a été accumulée un nombre de fois supérieur. Une fois toutes les courbes tracées (une courbe par point de l'image de contours) (figure 13), les cases avec les valeurs les plus élevées sont celles que nous retenons pour la transformée inverse (ce qui correspond aux plus longs contours).

La détection des points avec la valeur maximum est délicate. A cause de la discrétisation de l'espace de la transformée de Hough, l'intersection d'un très grand nombre de courbes crée une zone de points de forte valeur, et non un point isolé. Le seuil à partir duquel il faut retenir les points de la transformée est en conséquence difficile à trouver. En effectuant un seuillage simple, on a de plus beaucoup de parasites provoqués par des grands *nuages* de points qui sont l'accumulation d'un nombre élevé de courbes (figure 14*(b)*) ; ils représentent des points alignés par hasard mais n'ont pas de signification pour la perspective (ils sont souvent dus à la présence de surfaces fortement texturées dans la photographie originale).

Un traitement efficace contre ces problèmes est l'utilisation d'un filtre passe-haut de manière à mettre en valeur les points précis et effacer au maximum les nuages de points. Pour une zone avec des points plutôt précis (zones de quelques pixels) (figure 14*(a)*), les points importants sont bien mis en valeur (figure 14*(c)*). Par contre, pour un nuage de points qui n'a pas de réelle signification (figure 14*(b)*) et qui comporte un grand nombre d'accumulateurs de forte valeur génant le seuillage de la transformée, le traitement s'avère très efficace (figure 14*(d)*). Nous observons que les accumulateurs ont, après filtrage, une faible valeur. Le passe-haut utilisé est un filtre de convolution 2D avec le noyau 3×3 suivant :

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

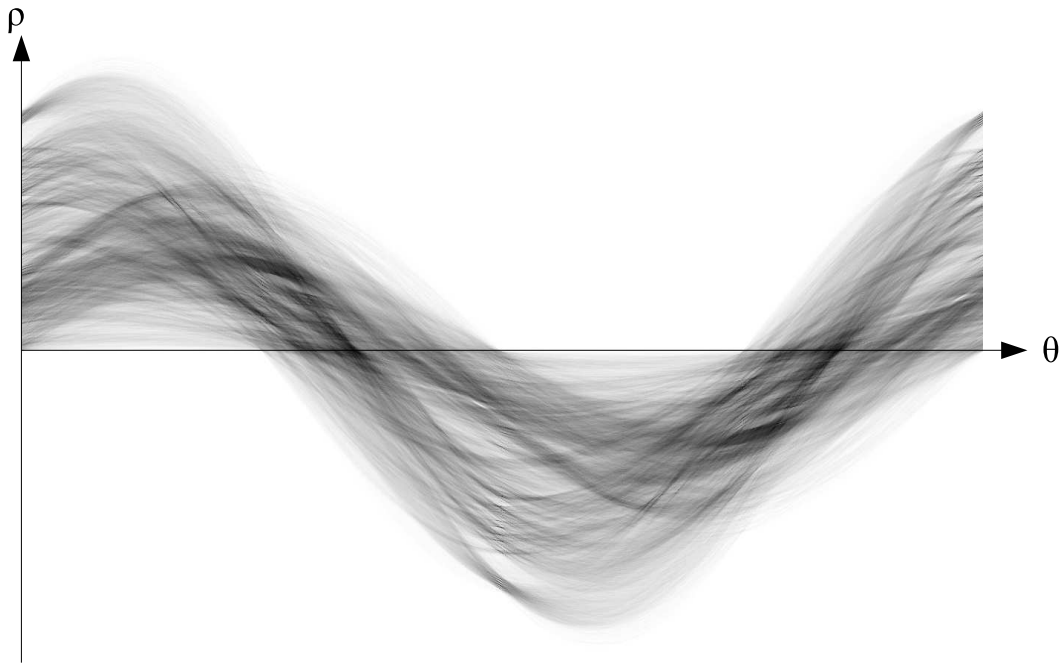

FIG. 13 – Résultat de la transformée de Hough pour 125734 points transformés sur une grille de 1620×1296 cases. Plus les points sont foncés, plus leur valeur est élevée.

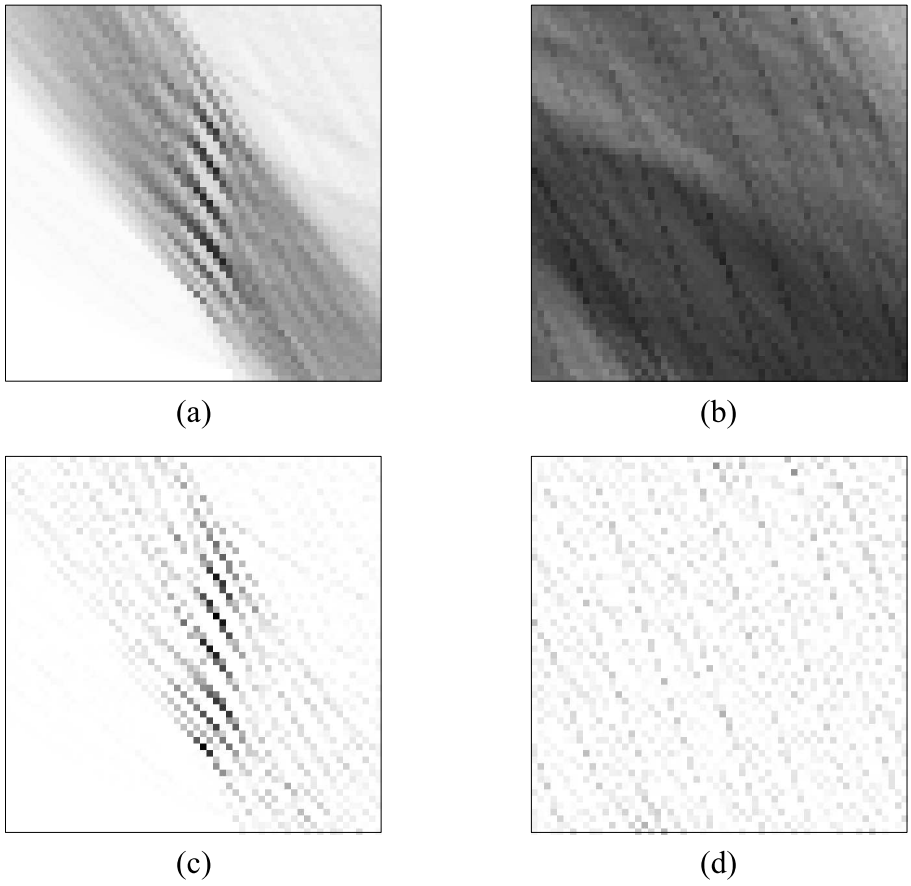

FIG. 14 – Effets du filtrage passe-haut sur la transformée de Hough (*(a)* zone avec des points à garder, *(b)* nuage de points sans signification, *(c)* zone avec des points à garder après filtrage, *(d)* nuage de points après filtrage)

Une fois ce traitement terminé, les accumulateurs dont la valeur dépasse un seuil déterminé manuellement sont listés. Les paramètres polaires (ρ, θ) sont retenus pour chaque droite résultante dans l'espace d'origine et vont servir pour la détection des points de fuite. La valeur du seuil a une influence sur la quantité de droites retenues : plus elle est basse, plus le nombre de droites à traiter est élevé, et plus le nombre de droites parasites est élevé.

2.2.5 Détection des points de fuite

Une fois la liste de droites obtenues (représentant les lignes de fuite), nous voulons trouver les points d'intersection de ces droites, nommés *points de fuite*. Les droites n'ayant pas d'intersections parfaites (ce sont plutôt des zones d'intersection), il est plus délicat de les trouver. De plus, les zones à trouver ne sont pas forcément à l'intérieur de l'image. Enfin, il n'y a pas toujours de *point d'intersection* à proprement parler, il peut aussi y avoir des directions qui sont la représentation de points de fuite infinis (voir la section 2.2.1).

Une méthode est proposée dans [7] qui consiste à utiliser l'image originale comme espace d'accumulation et d'y accumuler les lignes de fuite ; en cherchant le maximum, un point de fuite peut être trouvé. Cette méthode simple à implémenter possède plusieurs défauts : le principal est l'impossibilité de détecter les points de fuite en dehors de l'image (cas très courant) ; le second est la difficulté de détecter les points de fuite infinis. De plus, l'accumulateur de valeur maximum n'est pas forcément un point de fuite, il faut chercher plutôt une zone, mais ceci n'est pas évident.

Une autre méthode utilise la projection sur la *sphère gaussienne*, une variante de la transformée de Hough. Elle est décrite de plusieurs manières dans [10, 11, 12, 13, 7]. C'est la méthode que nous utilisons mais avec des améliorations concernant la précision du résultat et la vitesse d'exécution.

La première modification que nous réalisons est l'utilisation d'une hémisphère au lieu d'une sphère car les données seraient redondantes par symétrie. La méthode utilisée est une variante de la transformée de Hough, donc nous utilisons un espace d'accumulation qui est ici l'hémisphère (figure 15) que nous subdivisons. Le centre de l'hémisphère est le centre optique de la caméra ; son rayon est la moitié de la plus petite dimension de l'image en pixels (en général la hauteur).

Une droite de l'espace 3D et le centre de la caméra (point C) se trouvent dans un plan nommé *plan d'interprétation* ; l'intersection de ce plan avec le plan image forme la projection de la droite 3D qui est aussi une droite. L'intersection de ce plan avec une sphère gaussienne est un *grand cercle* ; nous obtenons un arc de cercle dans le cas d'une hémisphère. C'est cet arc de cercle qui sera accumulé sur l'hémisphère. Une fois toutes les droites accumulées, la présence de cellules de valeur élevée (qui représentent l'intersection d'un grand nombre de grand cercles) permet de mettre en évidence les points de fuite en projetant les accumula-

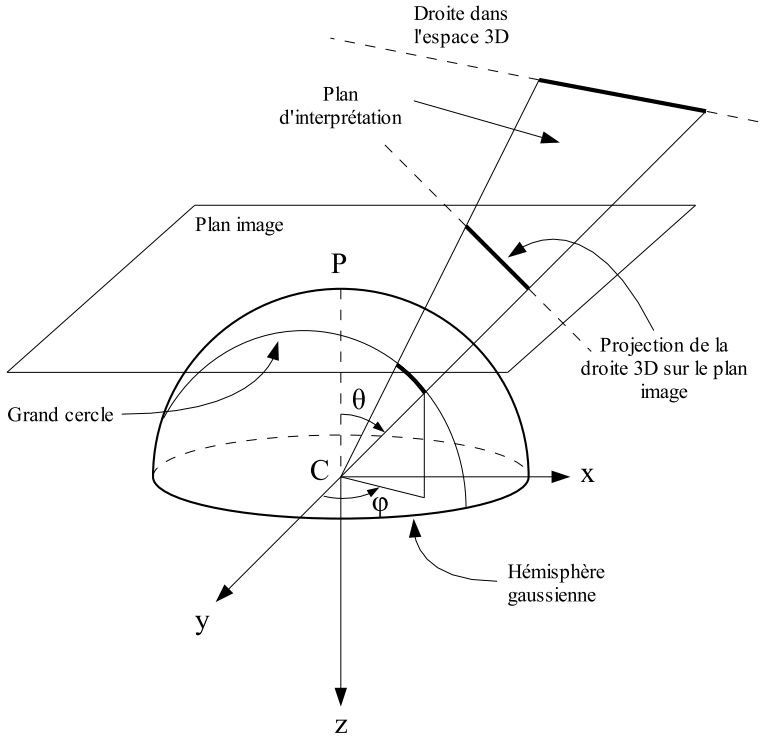

FIG. 15 – Hémisphère gaussienne et projection d'une ligne de fuite

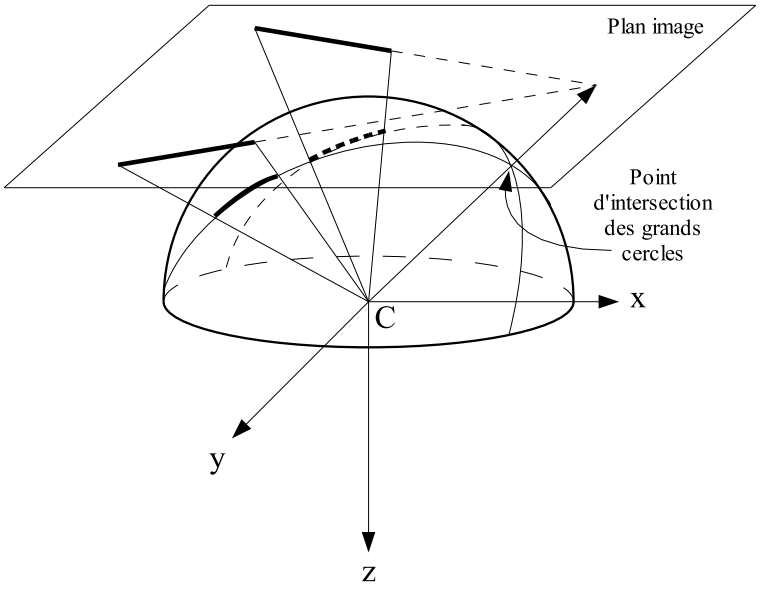

FIG. 16 – Détection d'un point de fuite

teurs de l'hémisphère gaussienne sur le plan image (figure 16).

Plusieurs subdivisions de la sphère gaussienne sont possibles :

- *subdivision régulière* : c'est la première qui a été proposée et c'est la plus simple à implémenter. Son grand inconvénient est la perte de précision quand le point de fuite à détecter est éloigné du centre de l'image car les paramètres φ et θ sont discrétisés régulièrement, menant donc à des accumulateurs d'aires différentes. Cette méthode est abordée dans [10, 11, 13, 7]. La projection des accumulateurs dans ce cas est illustrée sur la figure 17*(a)*, nous pouvons observer la grande densité d'accumulateurs projetés près du centre de l'image, mais une grande perte de précision a lieu dès que nous nous éloignons du centre à cause de la grande taille des projections.
- *subdivision adaptative* : elle permet de lutter contre les problèmes de précision en subdivisant un nombre de fois supérieur au niveau des grands cercles. Cette méthode a été proposée par [10] mais la présence d'un grand nombre de cercles (2164 dans le cas de la figure 13) a pour résultat une subdivision importante utilisant de plus des structures de données hiérarchiques assez lentes à parcourir.
- *subdivision hiérarchique* : le but est d'exécuter plusieurs fois la transformée de Hough sur la sphère gaussienne mais avec des résolutions de grilles régulières différentes. La méthode est citée dans [7]. Nous ne l'utilisons pas à cause du temps de calcul, et

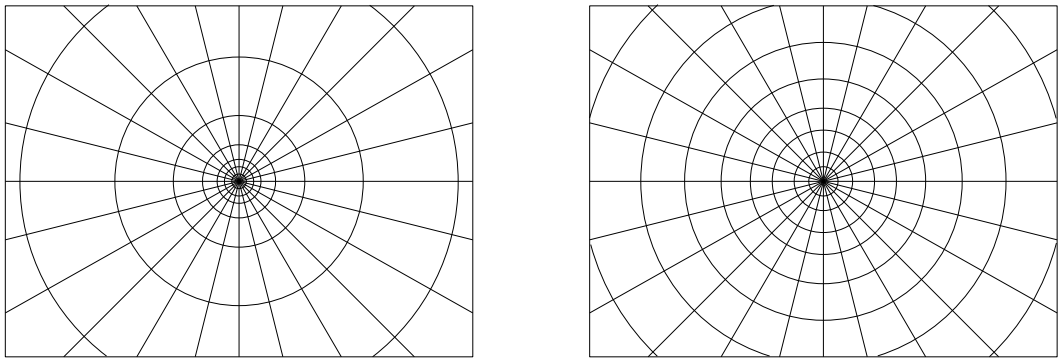

FIG. 17 – Projection des cellules de l'hémisphère gaussienne sur le plan image (*(a)* subdivision régulière, *(b)* subdivision régulière de φ et irrégulière de θ)

surtout à cause de la faible précision pour les points de fuite infinis due à l'utilisation de grilles régulières.

- *subdivision semi-régulière (quantification irrégulière de φ, quantification régulière de θ)* : cette méthode proposée par [13] introduit une subdivision efficace lors de l'exécution. Elle propose une bonne précision générale. La subdivision de φ change pour chaque θ, permettant d'obtenir une bonne précision pour la direction des points de fuite infinis et une précision inférieure près du centre de l'image. La subdivision régulière de θ présente un inconvénient : la précision de la distance des points de fuite par rapport au centre de l'image diminue quand la distance augmente ; cela est particulièrement génant pour les points de fuite finis. Cette situation est similaire à la subdivision régulière (figure 17*(a)*).
- *subdivision semi-régulière (quantification régulière de φ, quantification irrégulière de θ)* : cette méthode est inspirée de la précédente sauf que les subdivisions régulière et irrégulière sont inversées. La précision de la direction est constante peu importe la distance par rapport au centre de l'image, alors que la subdivision irrégulière de θ permet une adaptation de la précision, importante pour la distance des points de fuite finis au centre de l'image. Ceci est la méthode que nous retenons. Le résultat de la projection des accumulateurs de l'hémisphère sur la plan image est illustré sur la figure 17*(b)*, nous y observons que les cellules projetées ont une aire plus uniforme que dans le cas de la subdivision régulière.

Avec la méthode retenue, la précision près du centre de l'image est légèrement inférieure que dans le cas de la subdivision régulière, mais ce n'est pas un obstacle : si un point de fuite est détecté légèrement à côté de la réalité au niveau du pôle de l'hémisphère, la projection sur l'image originale ne sera décalée que de quelques pixels. Sur le reste de l'image,

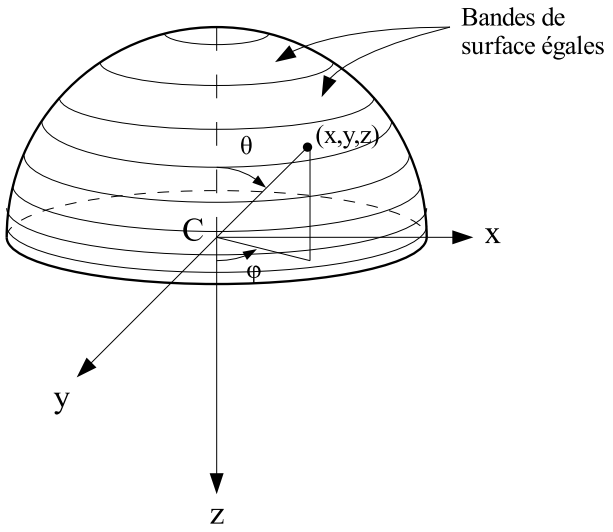

FIG. 18 – Paramétrage de l'hémisphère

les projections des cellules sont plus uniformes que dans le cas de la subdivision régulière, permettant de minimiser les écarts entre les différents points de fuite finis possibles. Cette uniformisation de la répartition permet d'obtenir un étalonnage de la caméra comportant moins d'erreurs importantes.

φ est subdivisé régulièrement :

$$\varphi_i = \frac{i}{N} \cdot 2\pi \qquad i \in \{0, \ldots, N-1\}$$

donc si les accumulateurs de l'hémisphère ont des aires égales, alors les *bandes* entourant l'hémisphère ont aussi des aires égales.

Introduisons une fonction $\Psi(\theta_i)$ qui n'est pas linéaire et qui possède la propriété suivante :

$$S(\theta_i) = K \cdot \Psi(\theta_i)$$

avec K une constante, et $S(\theta_i)$ la surface de la calotte sphérique pour $\theta \in [0, \theta_i]$. Nous avons défini que les bandes successives autour de l'hémisphère doivent avoir des aires égales, par conséquent la surface de la calotte sphérique est proportionnelle à $\Psi(\theta_i)$. Chaque indice i correspond à une bande, mais la relation liant l'angle θ_i à la surface de la calotte sphérique n'est pas linéaire, d'où la présence la fonction $\Psi(\theta_i)$ non linéaire. La figure 18 illustre tout ceci.

Nous voulons maintenant déterminer la fonction $\Psi(\theta_i)$. Pour cela, calculons l'aire de la calotte sphérique :

$$\begin{aligned} S(\theta_i) &= \int_0^{\theta_i}\int_0^{2\pi} \sin\theta \; d\varphi d\theta = \int_0^{2\pi} d\varphi \int_0^{\theta_i} \sin\theta \; d\theta \\ &= 2\pi[-\cos\theta]_0^{\theta_i} = 2\pi(1-\cos\theta_i) \\ &= K \cdot \Psi(\theta_i) \end{aligned}$$

Nous voulons $\Psi(\theta_i) \in [0,1]$ dans un but de simplification ; nous en déduisons donc :

$$\begin{cases} \Psi(\theta_i) = 1 - \cos\theta_i & \theta_i \in [0, \pi/2] \\ K = 2\pi \end{cases}$$

Nous pouvons remarquer que K correspond à la surface de la calotte sphérique pour l'hémisphère complète (c'est-à-dire pour $\theta = \pi/2$ et pour un rayon de 1 de l'hémisphère).

Nous voulons faire varier $\theta \in [0, \pi/2]$ de manière non uniforme de telle façon que $\Psi(\theta)$ varie uniformément :

$$\Psi(\theta_i) = \frac{i}{M} \qquad i \in \{0, \ldots, M-1\}$$

Maintenant que nous disposons d'un hémisphère $((\phi_j, \Psi(\theta_i)), j \in [0,N], i \in [0, M-1])$ correctement subdivisée, projetons les droites de l'image sur cette hémisphère. Un plan d'interprétation est défini par le centre optique C de la caméra et par une droite du plan image. Le centre de la caméra est considéré comme l'origine du monde, donc le plan d'interprétation passe toujours par cet origine, d'où la distance du plan par rapport à l'origine est toujours nulle ; le plan d'interprétation n'est donc représenté que par sa normale. Nous nous fixons une convention : la composante z de la normale est toujours positive ou nulle (par rapport au repère global), ceci est toujours possible. Un grand cercle est l'intersection d'un hémisphère et d'un plan, nous devons donc résoudre le système d'équations suivant (dans le reprère global) :

$$\begin{cases} n_x \cdot x + n_y \cdot y + n_z \cdot z = 0 \\ x^2 + y^2 + z^2 = r^2 \\ z \leq 0 \end{cases}$$

La première équation est celle du plan d'interprétation, avec (n_x, n_y, n_z) la normale à ce plan. La seconde équation est celle d'une sphère de rayon r. Enfin, la troisième, qui est une inéquation, permet de trouver l'intersection avec une hémisphère et non une sphère. En injectant la première équation dans la deuxième, nous obtenons :

$$(n_z^2 + n_x^2)x^2 + (n_z^2 + n_y^2)y^2 + n_x n_y xy = r^2 n_z^2$$

Nous voulons passer des coordonées cartésiennes aux coordonnées sphériques. Nous utilisons les équations de conversion suivantes :

$$\begin{cases} x = r\sin\theta\cos\varphi \\ y = r\sin\theta\sin\varphi \\ z = -r\cos\theta \end{cases}$$

En injectant ces trois équations dans la précédente, nous obtenons :

$$\sin^2\theta = \frac{n_z^2}{n_z^2 + (n_x\cos\varphi + n_y\sin\varphi)^2}$$

Or nous avons défini $n_z \geq 0$ et sachant que $\theta \in [0, \pi/2]$, alors $\sin\theta \geq 0$:

$$\sin\theta = \frac{n_z}{\sqrt{n_z^2 + (n_x\cos\varphi + n_y\sin\varphi)^2}}$$

L'équation finale de la courbe sur l'hémisphère gaussienne est, avec $\varphi \in [0, 2\pi]$:

$$\theta = C(\varphi) = \arcsin\left(\frac{n_z}{\sqrt{n_z^2 + (n_x\cos\varphi + n_y\sin\varphi)^2}}\right)$$

Une fois cette équation discrétisée, nous obtenons :

$$C(\varphi_i) = \arcsin\left(\frac{n_z}{\sqrt{n_z^2 + (n_x\cos\varphi_i + n_y\sin\varphi_i)^2}}\right) \qquad i \in \{0, \ldots, N-1\}$$

Il faut accumuler cette courbe sur l'hémisphère mais nous avons introduit une fonction $\Psi(\theta_i)$ permettant d'avoir une précision correcte en fonction de la distance par rapport au centre de l'image. Nous pouvons utiliser cette fonction et la formule $\cos(\arcsin(x)) = \sqrt{1-x^2}$ pour $x \in [0, 1]$; nous obtenons :

$$\begin{aligned}
\Psi(C(\varphi_i)) &= 1 - \cos\left(\arcsin\left(\frac{n_z}{\sqrt{n_z^2 + (n_x\cos\varphi_i + n_y\sin\varphi_i)^2}}\right)\right) \\
&= 1 - \sqrt{1 - \frac{n_z^2}{n_z^2 + (n_x\cos\varphi_i + n_y\sin\varphi_i)^2}} \\
&= 1 - \frac{n_x\cos\varphi_i + n_y\sin\varphi_i}{\sqrt{n_z^2 + (n_x\cos\varphi_i + n_y\sin\varphi_i)^2}} \\
&= 1 - \frac{t(n_x, n_y, \varphi_i)}{\sqrt{n_z^2 + t(n_x, n_y, \varphi_i)^2}}
\end{aligned}$$

Dans un but de simplification, nous retournons la définition de θ_i, c'est-à-dire qu'un angle de 0 correspond à l'équateur, et un angle de $\pi/2$ correspond au pôle. L'équation finale de la courbe dans l'espace d'accumulation est :

$$\Psi(C(\varphi_i)) = \frac{t(n_x, n_y, \varphi_i)}{\sqrt{n_z^2 + t(n_x, n_y, \varphi_i)^2}}$$

avec $t(n_x, n_y, \varphi_i) = n_x\cos\varphi_i + n_y\sin\varphi_i$.

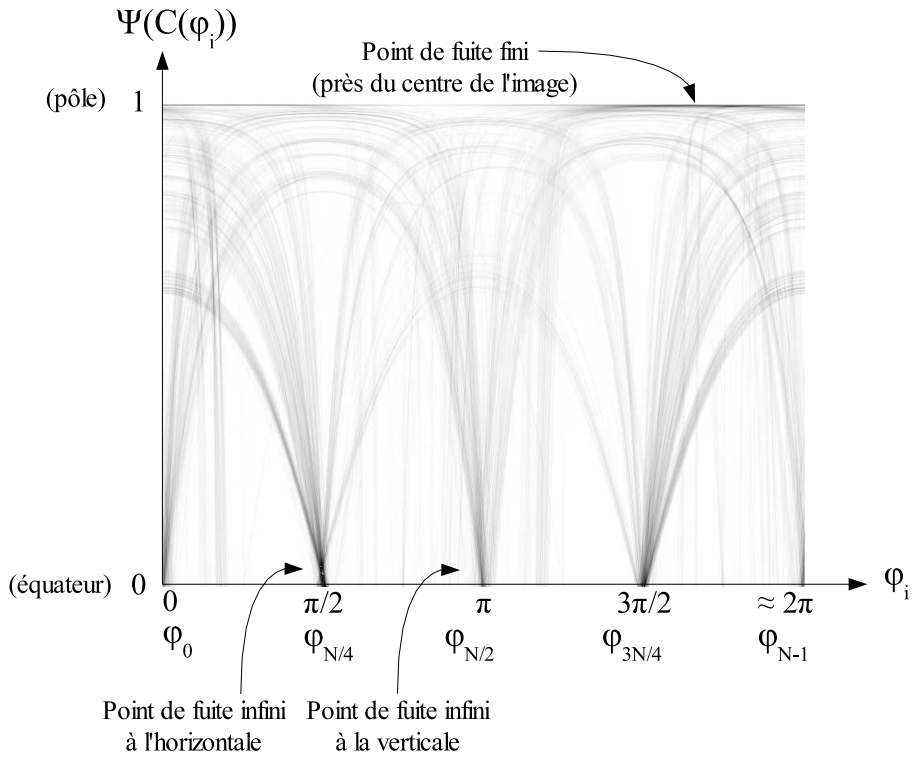

FIG. 19 – Résultat de la transformée de Hough sur hémisphère gaussienne pour 2164 droites transformées sur une grille de 800 cases pour φ et 600 cases pour θ . Plus les points sont foncés, plus leur valeur est élevée.

Sur la figure 19, nous pouvons observer le résultat de l'accumulation. Les dimensions utilisées, $N \times M = 800 \times 600$ sur cette figure, sont fixées manuellement par l'utilisateur, mais cette valeur par défaut donne de bons résultats dans le cas général. Une remarque concerne la détection des maxima dans cette transformée : la détection des cellules de valeur maximum renvoie des résultats erronés, imprécis ; ceci est dû à la taille des zones de forte accumulation. La méthode du filtre passe-haut utilisée pour la première transformée de Hough ne fonctionne pas ici car les courbes ne sont pas nettes, elles sont bruitées. Nous utilisons à l'inverse un filtre passe-bas qui réduit ce bruit. Le filtre de convolution 2D utilisé est le suivant (ici un noyau 5×5 mais la taille peut être paramétrée manuellement) :

$$\frac{1}{25}\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Une fois le filtre appliqué, il ne faut pas chercher les 3 cellules de plus forte valeur, car les résultats sont toujours erronés à cause de la taille des zones de forte accumulation. La technique que nous utilisons est la suivante :

```
faire 3 fois
{
  Appliquer un filtre passe-bas sur les accumulateurs ;
  Chercher la cellule de valeur maximum ;
  Parmi les droites projetées, chercher celles
      qui contribuent au point de fuite détecté ;
  Dans la transformée non filtrée, recommencer la
      projection de ces droites mais en soustrayant ;
}
```

Précisons que les points de fuite finis et infinis sont déterminés de la même façon. Plus précisémeent, dû à des problèmes de précision de calcul, un point de fuite infini est aussi un point d'intersection d'un ensemble de droites sur l'image. La différence est que ce point d'intersection se trouve très loin du point principal, donc pas sur l'image. On suppose qu'un point de fuite infini correspond à une valeur de $\theta = \frac{\pi}{2} - \epsilon$, ϵ étant une petite valeur.

Cette méthode se révèle relativement efficace ; de plus les droites sont classées en fonction de leurs valeurs accumulées, ce qui permet d'éliminer les droites parasites. Ceci est illustré sur la figure 20. Nous pouvons voir que le premier point de fuite fini est représenté par une croix rouge, les lignes de fuite qui ont contribué à la détection de ce point sont représentées en rouge foncé. Lors de la première boucle de l'algorithme précédent, les droites rouges sont reprojetées sur l'hémisphère gaussienne mais cette fois en soustrayant 1 aux accumulateurs de l'hémisphère. Une fois ceci réalisé, la deuxième boucle permet de trouver le point de fuite infini dont la direction est représentée par la ligne vert clair. Les lignes vert foncé ont permis de détecter cette direction et sont reprojetées sur l'hémisphère. Lors de la troisième boucle,

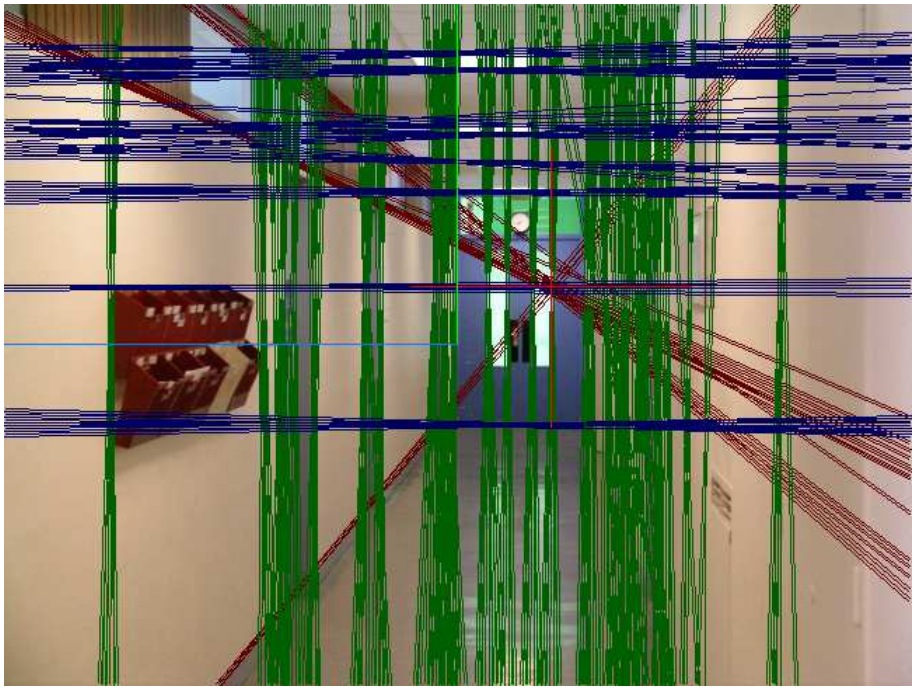

FIG. 20 – Contribution des lignes de fuite détectées pour chaque point de fuite (rouge : point de fuite fini, vert et bleu : points de fuite infinis.

le deuxième point de fuite infini est détecté, sa direction est représentée en bleu, et les lignes ayant contribué à la détection de cette direction sont représentées en bleu foncé. Les lignes parasites sont représentées en blanc, mais dans le cas de cette image, aucune n'est considérée comme telle.

Pour un point de fuite fini, le critère que nous retenons pour qu'une droite soit considérée comme contribuant à la détection du point de fuite est sa distance par rapport à ce point. Une distance de quelques pixels est un critère efficace mais est dépendant de la résolution de l'image originale et de la qualité de détection des lignes de fuite. Pour cela, une distance relative à la taille de l'image peut être utilisée (une valeur entre 0 et 1 fois la taille de l'image). Pour un point de fuite infini, l'angle entre la direction du point de fuite et la droite est un bon critère. Ce critère vaut quelques degrés dans le cas général.

2.3 Construction des données 3D

Une fois l'analyse de l'image terminée, un ensemble de trois points de fuite, finis ou infinis, est obtenu ; leurs coordonnées se situent dans le plan image. A partir de ces données, il faut retrouver les données 3D de manière à pouvoir déterminer les coordonnées des quadrilatères nécessaires pour le *Tour Into the Picture*. Dans un premier temps, les paramètres intrinsèques et la matrice de rotation de la caméra qui a servi à prendre la photographie originale doivent être déterminés, puis les coordonnées des quadrilatères, enfin les textures qui seront plaquées sur les quadrilatères.

2.3.1 Etalonnage de la caméra

La première étape de la reconstruction 3D est la détermination des paramètres intrinsèques et extrinsèques de la caméra, comme décrit dans la section 2.2.1. Nous fixons le centre optique de la caméra à l'origine du monde, soit $(0, 0, 0)$; c'est la scène qui est déplacée et non la caméra. La focale est d'abord déterminée pour en déduire ensuite l'orientation de la caméra. Les trois combinaisons possibles de points de fuite finis et infinis introduisent des raisonnements différents. Notons que le repère (u, v, w) est le repère du monde, où u et v correspondent respectivement aux axes horizontal et vertical. Le repère (x, y, z) est celui de la caméra.

Un point de fuite fini, deux points de fuite infinis :

Cette situation est représentée sur la figure 21. $F(f_x, f_y, -f)$ est le point de fuite fini, $\vec{I_1} = (I_{1x}, I_{1y}, 0)$ et $\vec{I_2} = (I_{2x}, I_{2y}, 0)$ sont les directions des points de fuite infinis dans le plan image. Ces vecteurs peuvent être normalisés mais ce n'est pas une obligation pour la suite des calculs. Un problème se pose concernant la focale : dans plusieurs documents, il est dit que la focale doit être spécifiée manuellement car l'image n'apporte pas assez d'informations pour l'étalonnage, ceci est confirmé par [2]. Dans notre cas, cela va être possible par la présence des vecteurs directeurs des deux points de fuite infinis. Le calcul de la matrice

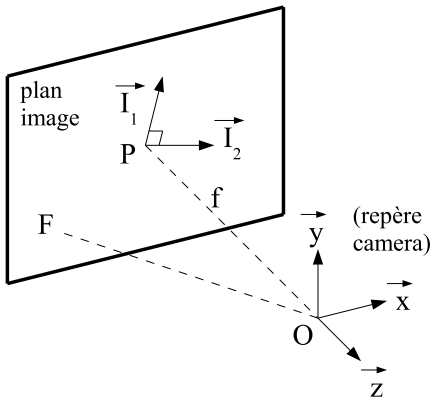

FIG. 21 – Etalonnage de la caméra avec un point de fuite fini et deux points de fuite infinis

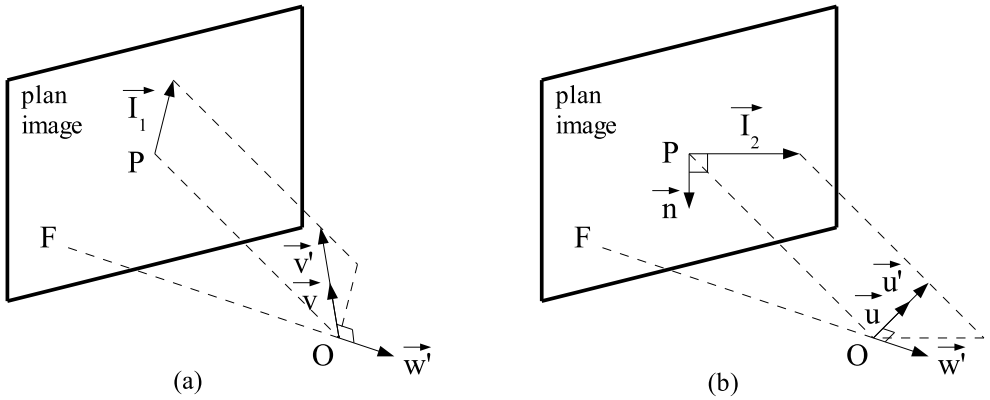

FIG. 22 – Détermination des vecteurs composant la matrice de rotation de la caméra

de rotation $(\vec{u}\ \ \vec{v}\ \ \vec{w})$ est d'abord réalisé pour en déduire finalement la focale f. La matrice de rotation est une matrice de passage de l'espace de la caméra vers l'espace global, ses colonnes sont les vecteurs de la base de la caméra dans le repère global.

Un premier vecteur non normalisé de la matrice de rotation peut être déduit du point de fuite fini (dans le repère global) :

$$\vec{w'} = (w'_x, w'_y, w'_z) = -\overrightarrow{OF} = (-f_x, -f_y, f)$$

Calculons maintenant $\vec{v'}$. Ce vecteur est indiqué sur la figure 22*(a)*, il se trouve dans le plan défini par les points O, P et la direction $\vec{I_1}$. Le vecteur $\vec{v'}$ est en fait le vecteur $\vec{I_1}$ auquel

une composante suivant l'axe $\vec{z}$ est ajoutée, donc $\vec{v'} = (I_{1x}, I_{1y}, v'_z)$. Nous voulons que les vecteurs $\vec{v'}$ et $\vec{w'}$ appartiennent à un trièdre orthogonal dans le but de former une base, nous voulons donc $\vec{v'} \cdot \vec{w'} = 0$. Cela conduit donc à :

$$I_{1x}w'_x + I_{1y}w'_y + v'_z w'_z = 0$$

$$v'_z = -\frac{I_{1x}w'_x + I_{1y}w'_y}{w'_z}$$

$$v'_z = \frac{I_{1x}f_x + I_{1y}f_y}{f}$$

Le troisième vecteur de la base, $\vec{u'}$, comme indiqué sur la figure 22*(b)*, est déterminé en sachant que nous voulons obtenir un trièdre direct et que $\vec{u'}$ doit être dans le plan formé par les points O, P et la direction $\vec{I_2}$:

$$\vec{u'} = \vec{v'} \wedge \vec{w'} = \begin{pmatrix} I_{1x} \\ I_{1y} \\ \frac{I_{1x}f_x + I_{1y}f_y}{f} \end{pmatrix} \wedge \begin{pmatrix} -f_x \\ -f_y \\ f \end{pmatrix} = \begin{pmatrix} I_{1y}f + \frac{I_{1x}f_x + I_{1y}f_y}{f} f_y \\ -\frac{I_{1x}f_x + I_{1y}f_y}{f} f_x - I_{1x}f \\ -I_{1x}f_y + I_{1y}f_x \end{pmatrix}$$

$\vec{n}$ est la normale au plan formé par les points O, P et la direction $\vec{I_2}$, $\vec{n} = (I_{2y}, -I_{2x}, 0)$. Nous voulons $\vec{u'} \cdot \vec{n} = 0$, nous obtenons donc, sachant que nous voulons une focale positive :

$$f = \sqrt{\left| \frac{(I_{1x}f_x + I_{1y}f_y)(I_{2x}f_x + I_{2y}f_y)}{I_{1x}I_{2x} + I_{1y}I_{2y}} \right|}$$

Connaissant la focale f, nous pouvons maintenant obtenir les vecteurs $\vec{u'}$, $\vec{v'}$ et $\vec{w'}$. Les vecteurs $\vec{u}$, $\vec{v}$ et $\vec{w}$ sont les versions normalisées des vecteurs précédents :

$$\vec{u} = \frac{\vec{u'}}{\|\vec{u'}\|} \qquad \vec{v} = \frac{\vec{v'}}{\|\vec{v'}\|} \qquad \vec{w} = \frac{\vec{w'}}{\|\vec{w'}\|}$$

Lors de l'expérimentation, nous nous apercevons que les résultats manquent de précision, le calcul de la focale renvoie par exemple 4.43 (soit un champ de vision de 25.4 °). Lors de l'édition des polygônes, nous obtenons des résultats aberrants par rapport à l'image originale. Si nous fixons la focale à 2.24 (soit un champ de vision de 48 ° qui est une valeur courante), les résultats sont bien meilleurs. Ce problème de précision provient de l'angle entre $\vec{I_1}$ et $\vec{I_2}$: cet angle est presque orthogonal et pour obtenir $\vec{u}$, $\vec{v}$ et $\vec{w}$ orthogonaux, la focale doit être modifiée de manière conséquente dès la moindre variation d'angle entre $\vec{I_1}$ et $\vec{I_2}$.

Pour pallier ce problème, nous fixons manuellement le champ de vision (et donc la focale). D'une image à l'autre, le champ de vision change mais reste toujours autour d'une valeur :

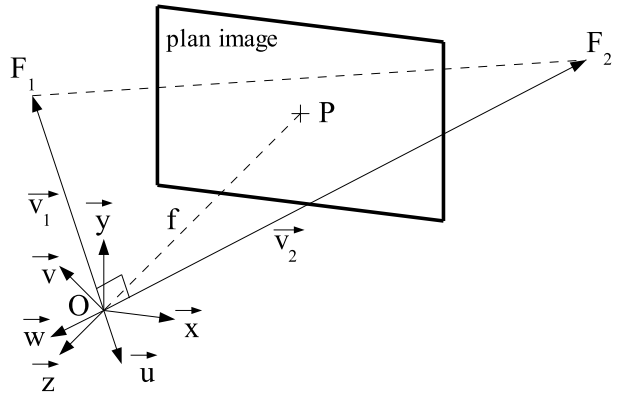

FIG. 23 – Etalonnage de la caméra avec deux points de fuite finis et un point de fuite infini

48 °. L'erreur de focale qui en découle n'est pas très importante et est à peine perceptible ; si des mesures devaient être réalisées sur l'image, des erreurs de proportions entre l'avant-plan et l'arrière-plan se ressentiraient, mais l'usage principal du *Tour Into the Picture* est la promenade en 3D en temps réel, et dans ce cas les déformations des textures sont à peine perceptibles.

Au niveau des calculs, peu de choses changent. La focale f est connue, donc $\vec{w'}$ est connu dès le début. Le vecteur $\vec{v'}$ se déduit aussi très facilement. Le vecteur $\vec{u'}$ est calculé par le produit vectoriel $\vec{u'} = \vec{v'} \wedge \vec{w'}$, il fait partie d'un trièdre direct mais ne se trouve pas forcément dans le plan formé par O, P et $\vec{I_2}$. Ceci est dû à la fixation manuelle de la focale et nous avons privilégié la précision pour les verticales plutôt que pour les horizontales de l'image, car elles sont en général plus nombreuses et mieux définies (dans les images comportant beaucoup de bâtiments en particulier, et aussi dans les images de forêt par exemple). Enfin les trois vecteurs peuvent être normalisés pour former la matrice de rotation.

Deux points de fuite finis, un point de fuite infini :

Cette situation est représentée sur la figure 23. Une méthode de calcul est proposée dans [14] mais nous en utilisons une autre qui demande moins de calculs, donc comporte moins de perte de précision lors de l'exécution. Dans [15], une méthode est proposée, similaire à la notre, mais formulée autrement.

Deux points de fuite finis (F_1 et F_2) signifient que deux vecteurs de la matrice de rotation sur trois ne sont pas dans le plan image, il s'agit de $\vec{v_1}$ et de $\vec{v_2}$. Un point de fuite infini signifie que le troisième vecteur de la matrice de rotation est dans le plan image. Ceci a pour conséquence que le plan défini par le triangle OF_1F_2 est orthogonal au plan image et donc

finalement que la droite (F_1F_2) doit passer par le point P. Or ce n'est pas le cas, comme indiqué sur la figure 23. Cela signifie que le troisième vecteur de la matrice de rotation n'est pas dans le plan image et donc qu'il y a un troisième point de fuite fini mais qui se trouve loin du point principal ; ce qui explique pourquoi il a été détecté comme un point de fuite infini dans l'étape de détection des trois points de fuite. Nous nous servons donc uniquement de F_1 et de F_2 pour les calculs, une partie du résultat est calculée par déduction.

Nous utilisons une hypothèse forte qui est que les trois points de fuite détectés représentent des directions orthogonales dans l'espace. Les vecteurs $\vec{v_1}$ et $\vec{v_2}$ sont donc considérés comme orthogonaux et nous pouvons en déduire la focale. Soit

$$\begin{cases} F_1(f_{1x}, f_{1y}, -f) \\ F_2(f_{2x}, f_{2y}, -f) \end{cases}$$

Nous en déduisons :

$$\begin{cases} \vec{v_1} = \overrightarrow{OF_1} = (f_{1x}, f_{1y}, -f) \\ \vec{v_2} = \overrightarrow{OF_2} = (f_{2x}, f_{2y}, -f) \end{cases}$$

Nous voulons $\vec{v_1} \cdot \vec{v_2} = 0$, donc $f_{1x}f_{2x} + f_{1y}f_{2y} + f^2 = 0$.

$$f^2 = -(f_{1x}f_{2x} + f_{1y}f_{2y})$$

$$f = \pm\sqrt{-(f_{1x}f_{2x} + f_{1y}f_{2y})}$$

Nous voulons obtenir une focale f positive donc l'expression finale de la focale est :

$$f = \sqrt{|f_{1x}f_{2x} + f_{1y}f_{2y}|}$$

Une fois la focale trouvée, nous pouvons déterminer la matrice de rotation $(\vec{u} \quad \vec{v} \quad \vec{w})$. Cette matrice est obtenue simplement :

$$\vec{u} = -\frac{\overrightarrow{OF_1}}{\|\overrightarrow{OF_1}\|} \qquad \vec{w} = -\frac{\overrightarrow{OF_2}}{\|\overrightarrow{OF_2}\|} \qquad \vec{v} = \vec{w} \wedge \vec{u}$$

Trois points de fuite finis :

L'étalonnage d'une caméra à partir de 3 points de fuite finis est un problème comportant trop de données. Les 3 points ne représentent pas forcément 3 directions parfaitement orthogonales, mais nous faisons l'hypothèse qu'elles le sont presque (montré sur la figure 24). La technique classique est la minimisation d'une fonction par des méthodes analytiques, en utilisant des opérations comme la SVD (Singular Value Decomposition) ; ce type de méthode est décrit dans [16]. Nous avons décidé d'utiliser une méthode plus simple et qui doit donner de meilleurs résultats. Le principe est de chercher un trièdre orthogonal qui se sert de deux points de fuite de manière exacte, puis du troisième pour calculer un facteur de qualité

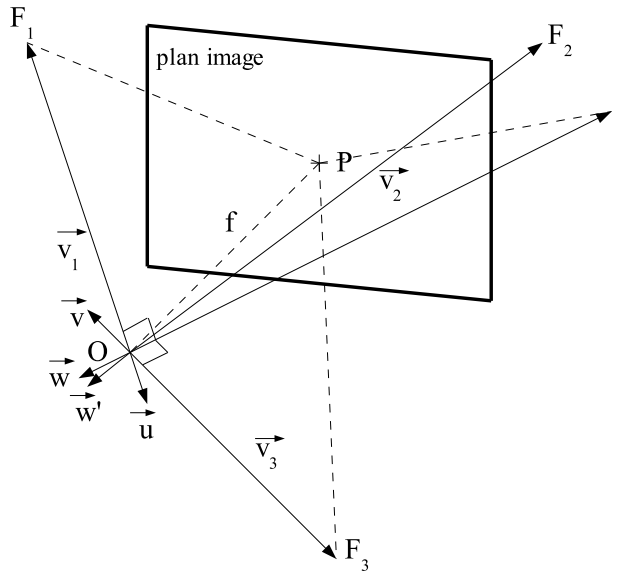

FIG. 24 – Exemple d'étalonnage de la caméra avec trois points de fuite finis

que nous voulons maximiser. L'exactitude des deux premières directions donne des résultats visuellement satisfaisants, une seule direction est incorrecte. Si nous utilisons la méthode de minimisation analytique, l'erreur finale peut être plus faible, mais le trièdre ne serait aligné avec aucun point de fuite, ce qui serait plus désagréable visuellement. Un exemple de résultat désiré est illustré sur la figure 24. L'algorithme simplifié est le suivant :

```
Pour chaque triplet (u,v,w) de directions de points de fuite
{
  Calculer la focale f en utilisant u et v ;
  Déterminer w' par produit vectoriel de u et v (base orthogonale) ;
  Calculer le facteur de qualité (produit scalaire de w et w') ;
}
Déterminer le plus grand facteur de qualité ;
Réordonner et normaliser les vecteurs du triplet correspondant ;
```

Il existe 6 triplets possibles de directions de points de fuite :

$$(\vec{v_1},\vec{v_2},\vec{v_3}),\ (\vec{v_1},\vec{v_3},\vec{v_2}),\ (\vec{v_2},\vec{v_3},\vec{v_1}),\ (\vec{v_2},\vec{v_1},\vec{v_3}),\ (\vec{v_3},\vec{v_1},\vec{v_2}),\ (\vec{v_3},\vec{v_2},\vec{v_1}).$$

Mais des symétries interviennent et seuls 3 de ces triplets ont une utilité, nous retenons :

$$(\vec{v_1},\vec{v_2},\vec{v_3}),\ (\vec{v_2},\vec{v_3},\vec{v_1}),\ (\vec{v_3},\vec{v_1},\vec{v_2}).$$

La focale est calculée de la même manière qu'avec 2 points de fuite finis, en utilisant les deux première directions :

$$f = \sqrt{|f_{1x}f_{2x} + f_{1y}f_{2y}|}$$

f_{1x}, f_{1y}, f_{2x} et f_{2y} étant les composantes des deux premières directions ($\vec{v_1}$ et $\vec{v_3}$ dans le cas de la figure 24).

Plusieurs techniques sont possibles pour la détermination du facteur de qualité, nous avons choisi le produit scalaire entre la direction du troisième point de fuite et la direction que nous souhaitons. Ces deux directions sont normalisées avant d'être utilisées dans le produit scalaire de manière à obtenir le cosinus de l'angle entre les deux vecteurs.

Le plus grand facteur de qualité parmi les trois combinaisons de points de fuite représente le trièdre othogonal s'approchant le plus des points de fuite de l'image. Le trièdre est normalisé de manière à être orthonormal.

2.3.2 Editeur de polygônes

L'analyse de l'image originale permet de déterminer les paramètres de la caméra. Par contre, à partir d'une seule image, il n'est pas possible de connaître la structure réelle de la scène. La projection du monde 3D sur un plan 2D provoque la perte d'un grand nombre d'informations et les retrouver est une tâche quasiment impossible. Des à priori peuvent être utilisés de manière à retrouver des éléments de la scène, mais dans notre cas, nous ne possédons quasiment aucun indice à l'exception de l'orientation de la caméra et de sa focale. Le principe du *Tour Into the Picture* expliqué dans [1, 2, 3, 4] est d'utiliser un parallélépipède pour approcher la forme de la scène. La reconstruction n'est bien sûr pas exacte, mais les résultats sont convaincants quand les mouvements dans la scène reconstruite n'ont pas une amplitude trop importante. Malheuseusement l'utilisateur doit paramétrer la forme de la scène manuellement. Ce travail de l'utilisateur est long, et impose des contraintes sur l'image originale (au niveau de l'alignement avec les bords de l'image par exemple). Comme l'analyse de l'image originale est incapable de *comprendre* la topologie de la scène, l'utilisateur doit préciser manuellement les limites des quadrilatères définissant la scène 3D. Avec notre approche, peu de paramètres doivent être fournis, 5 en l'occurrence : ce sont les distances des 5 quadrilatères par rapport au centre optique de la caméra (*Left*, *Right*, *Floor*, *Ceiling* et *Back*). Comme la caméra est déjà étalonnée, la rotation n'a pas à être fournie par l'utilisateur, tout comme l'effet de profondeur créé par la focale.

L'interface que nous retenons est illustrée sur les figures 25 et 26. Pour régler les dimensions des quadrilatères, 4 poignées doivent être déplacées en faisant glisser la souris, elles modifient les paramètres *Left wall*, *Right wall*, *Floor* et *Ceiling*. Chaque poignée agit sur 2 paramètres simultanément, 2 poignées suffisent mais dans un cas comme la figure 26, une seule poignée serait affichée (l'autre serait en dehors de l'écran). A l'inverse, la distance de l'arrière-plan à la caméra ne peut être réglée que par le champ de saisie correspondant ; ce

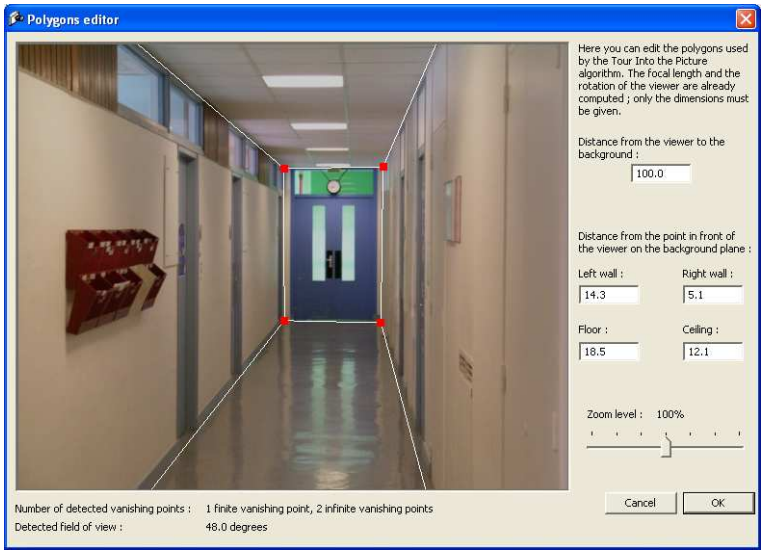

FIG. 25 – Editeur de polygônes (couloir avec un point de fuite fini)

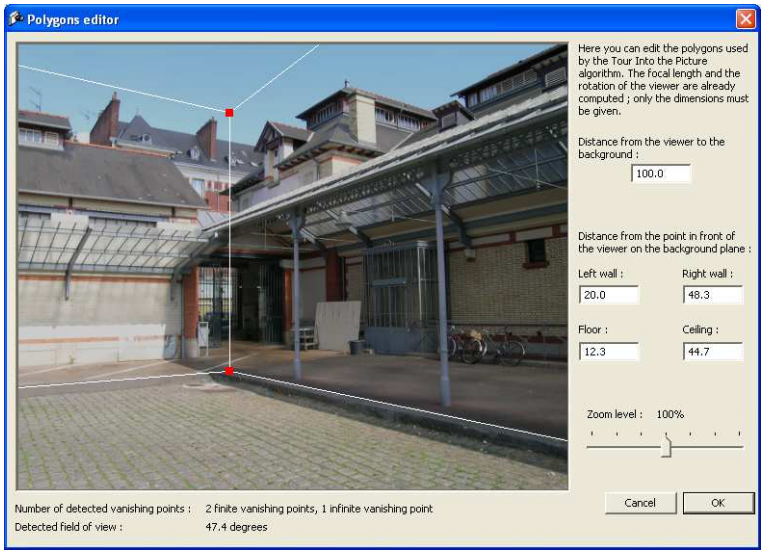

FIG. 26 – Editeur de polygônes (halles avec deux points de fuite finis)

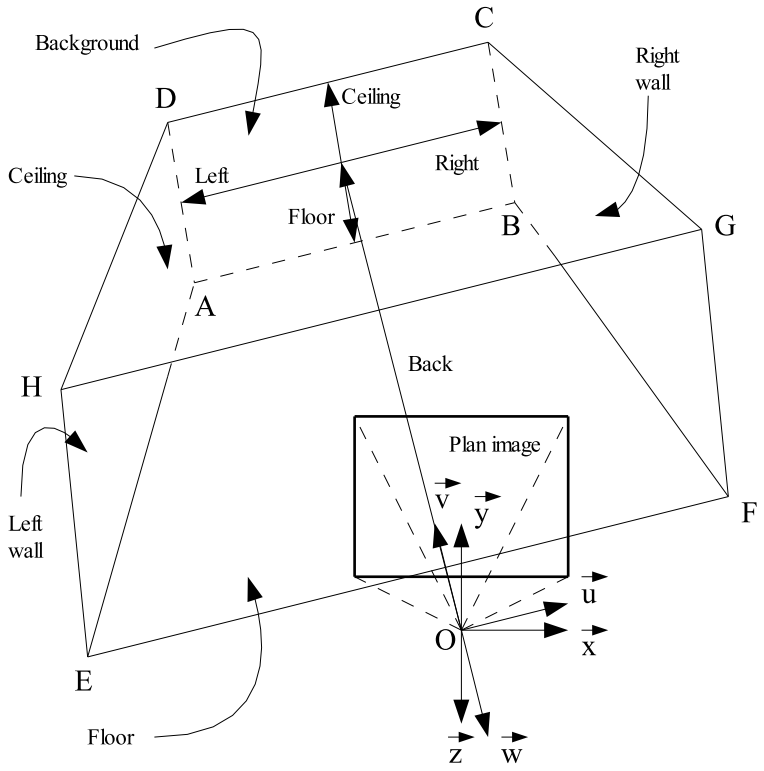

FIG. 27 – Les 5 quadrilatères définissant la scène et leurs paramètres

paramètre a une influence sur les dimensions de la scène et donc sur les 4 autres paramètres, il vaut mieux donc régler cette distance en premier puis ajuster le parallélépipède à la souris. Pour améliorer la précision de l'ajustement, la possibilité de zoomer est offerte de manière à agrandir ou réduire la vision de l'image.

Les valeurs initiales des 5 paramètres sont fixées arbitrairement, par des constantes : 100 pour *Back*, 20 pour *Left*, *Right*, *Floor* et *Ceiling*. Ces valeurs permettent d'obtenir des poignées visibles dans la fenêtre de l'éditeur de polygônes pour la plupart des photographies, permettant de réduire l'utilisation de la fonction de zoom. Ces paramètres pourraient être estimés au lieu d'être fixés à une constante, mais ceci demande des travaux supplémentaires au niveau de l'analyse de l'image originale de manière à approcher au mieux le parallélépipède de la forme de la scène. Ces calculs n'apporteraient qu'une estimation et l'intervention de l'utilisateur serait toujours nécessaire, alors fixer les valeurs initiales à des constantes est un gain de temps, aussi bien au niveau du développement que de l'exécution.

Une fois les 5 paramètres déterminés, les coordonnées des sommets des 5 quadrilatères sont calculées dans le repère du monde comme suit (sachant que $Left$, $Right$, $Floor$, $Ceiling$ et $Back$ sont des nombres positifs) :

$$\begin{array}{lcccccc}
\overrightarrow{OA} = & - & Left \cdot \vec{u} & - & Floor \cdot \vec{v} & - & Back \cdot \vec{w} \\
\overrightarrow{OB} = & & Right \cdot \vec{u} & - & Floor \cdot \vec{v} & - & Back \cdot \vec{w} \\
\overrightarrow{OC} = & & Right \cdot \vec{u} & + & Ceiling \cdot \vec{v} & - & Back \cdot \vec{w} \\
\overrightarrow{OD} = & - & Left \cdot \vec{u} & + & Ceiling \cdot \vec{v} & - & Back \cdot \vec{w} \\
\overrightarrow{OE} = & - & Left \cdot \vec{u} & - & Floor \cdot \vec{v} & & \\
\overrightarrow{OF} = & & Right \cdot \vec{u} & - & Floor \cdot \vec{v} & & \\
\overrightarrow{OG} = & & Right \cdot \vec{u} & + & Ceiling \cdot \vec{v} & & \\
\overrightarrow{OH} = & - & Left \cdot \vec{u} & + & Ceiling \cdot \vec{v} & &
\end{array}$$

2.3.3 Calcul des textures

Les quadrilatères calculés précédemment représentent la topologie approximative de la scène. En plaquant des textures sur ces quadrilatères, nous pouvons obtenir un monde approximatif mais qui peut être convaincant. Une texture par quadrilatère est donc nécessaire, mais dans un soucis de consommation mémoire, seules les textures utiles sont calculées, les quadrilatères n'apparaissant pas dans l'image originale ne sont pas pris en compte. Les textures obtenues pour l'image de la figure 28*(a)* sont représentées sur les figures 28 *(b)*,*(c)*,*(d)*,*(e)* et *(f)*.

Pour déterminer la résolution des différentes textures, deux paramètres doivent être pris en compte :

- la taille finale du fichier exporté vers *ATIP navigator* : plus la résolution des textures est élevée, plus la qualité finale est bonne, mais plus le fichier exporté est volumineux. A l'inverse, si la résolution des textures est faible, le fichier exporté est plus petit, par exemple pour l'utilisation sur un PDA, mais la qualité est inférieure. Seul l'utilisateur peut spécifier à quelle plateforme est destinée le fichier exporté, il a donc à choisir dans *ATIP maker* la résolution maximum des textures. L'effet du choix de la résolution est illustré sur la figure 29.
- la taille de la projection des quadrilatère sur l'image originale : si une ou plusieurs des projections ont une aire faible sur l'image originale, peu de données sont fournies pour calculer les textures, il est donc inutile d'utiliser une résolution trop élevée.

Nous utilisons donc une méthode semi-automatique qui prend en compte ces deux critères : résolution maximum et aire de la projection (illustré sur la figure 28*(b)* qui est une texture plus petite que les autres car l'aire recouverte sur l'image originale est plus faible). Pour optimiser la taille du fichier exporté, les images peuvent être compressées avec un algorithme destructif comme *JPEG*. Cette compression est relativement efficace car les textures calculées contiennent beaucoup de grandes zones uniformes noires qui représentent les zones

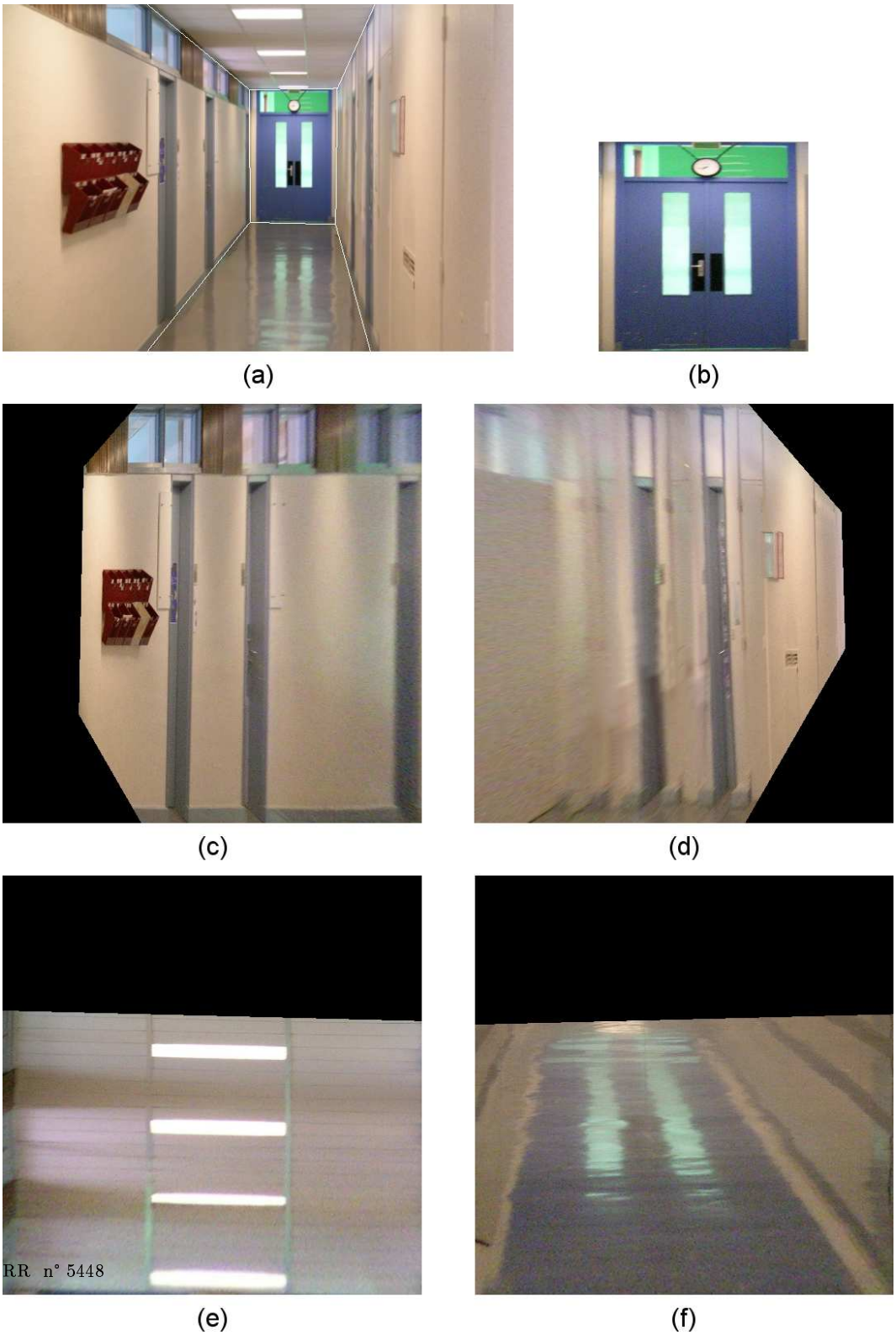

FIG. 28 – Textures calculées pour la scène du couloir (*(a)* image originale avec affichage des bordures des quadrilatères, *(b)* texture du fond (512×512), *(c)* texture du mur gauche (1024×1024), *(d)* texture du mur droit (1024×1024), *(e)* texture du plafond (1024×1024), *(f)* texture du sol (1024×1024))

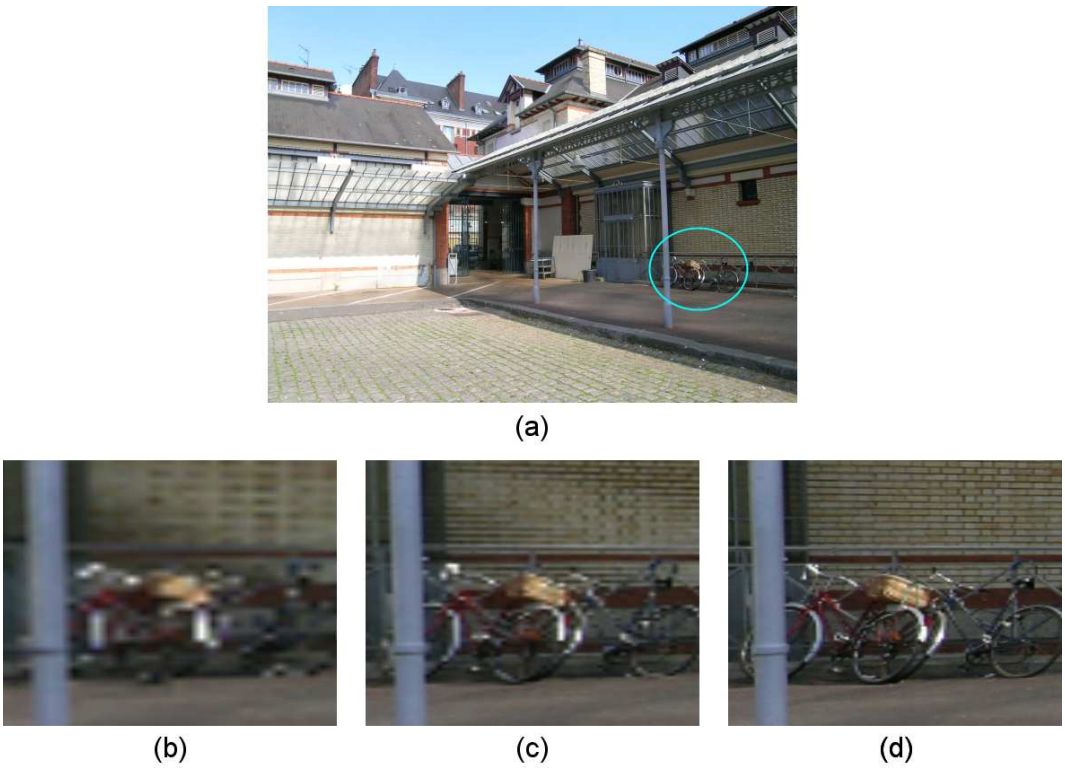

FIG. 29 – Influence de la taille des textures sur la qualité finale (*(a)* image originale avec indication de la zone observée, *(b)* vue rapprochée, taille maximum des textures : 256, la texture est floue, *(c)* taille maximum des textures : 512, la qualité est satisfaisante mais du moiré est encore présent, *(d)* taille maximum des textures : 1024, la qualité est bonne pour un moniteur PC)

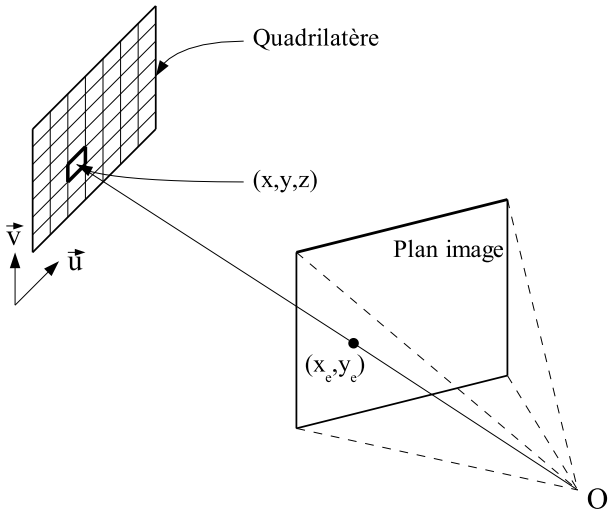

FIG. 30 – Projection d'un texel de coordonnées (u, v) sur l'image originale pour en déterminer la couleur. (x, y, z) est la coordonnées dans l'espace du texel, (x_e, y_e) sa projection sur l'image

inconnues de l'image originale (figure 28 *(c),(d),(e),(f)*).

Le calcul d'une texture s'effectue ainsi : le quadrilatère correspondant dans l'espace 3D est discrétisé en une grille 2D de *texels*, chacun est projeté sur le plan de l'image originale. Les coordonnées obtenues sont des nombres réels ; pour trouver la couleur résultante extraite de l'image originale, une interpolation bilinéaire est utilisée. L'illustration de cette méthode est donnée sur la figure 30.

2.3.4 Exportation des données 3D

Une fois tous les traitements sur l'image terminés, les données doivent être exportées de manière à être utilisables avec un navigateur. *ATIP navigator* est la version PC sous Windows du navigateur, il permet de profiter de la haute résolution des moniteurs pour offrir une meilleure qualité d'image. Le navigateur peut aussi être implémenté sur un PDA, où la résolution doit être inférieure. Ces contraintes sont prises en compte dans le format du fichier à exporter. Ce fichier contient tous les renseignements nécessaires à la navigation, les coordonnées, les textures compressées, etc....Le format du fichier est extensible : il est constitué de blocs avec un identificateur unique. De nouveaux blocs peuvent être créés tout en gardant la compatibilité avec les anciennes versions du fichier car les blocs inconnus sont ignorés à la lecture et remplacés par les valeurs par défaut. Les spécifications du fichier sont les suivantes :

```
* fichier ATIP :  - identificateur "ATIPFILE" (8 octets)
```

```
                - ensemble de blocs consécutifs

* bloc :        - identificateur ID (2 octets)
                - données dépendant du bloc

* bloc ATIP_CHUNK_ORIGINAL_IMAGE_SIZE (taille de l'image originale) :
                - ID = 0x0100
                - largeur en pixels (4 octets)
                - hauteur en pixels (4 octets)

* bloc ATIP_CHUNK_VERTICES (coordonnées des sommets pour le
                          parallélépipède aligné avec les axes) :
                - ID = 0x0200
                - nombre de sommets (4 octets) (=8 pour le moment)
                - coordonnées des sommets
                    (nombre sommets * 3 * 4 octets (floats))

* bloc ATIP_CHUNK_ROTVERTICES (coordonnées des sommets pour le
                             parallélépipède à afficher) :
                - ID = 0x0201
                - nombre de sommets (4 octets) (=8 pour le moment)
                - coordonnées des sommets
                    (nombre sommets * 3 * 4 octets (float))

* bloc ATIP_CHUNK_INDICES (indices des sommets pour chaque sommet
                          de chaque quadrilatère) :
                - ID = 0x0210
                - nombre de quadrilatères (4 octets) (=5 pour le moment)
                - indices des sommets de chaque quadrilatère
                    (nombre quadrilatères * 4 * 4)

* bloc ATIP_CHUNK_POLYGONS_PARAMETERS (paramètres ayant servi à définir
                                      les dimensions de la scène) :
                - ID = 0x0220
                - distance de la caméra au quadrilatère du fond
                    (4 octets, float)
                - distance de la caméra au quadrilatère gauche
                    (4 octets, float)
                - distance de la caméra au quadrilatère droit
                    (4 octets, float)
                - distance de la caméra au quadrilatère du sol
                    (4 octets, float)
```

```
                - distance de la caméra au quadrilatère du plafond
                    (4 octets, float)

* bloc ATIP_CHUNK_TEXTURE (données de texture pour un quadrilatère) :
                - ID = 0x0300
                - identificateur de la texture
                    (>=0 et <=4 pour le moment) (4 octets)
                - taille des données en octets (4 octets)
                - données de la texture (n'importe quel format géré
                                         par la bibliothèque DevIL)

* bloc ATIP_CHUNK_FOCAL_FOV (paramètres intrinsèques de la caméra) :
                - ID = 0x0400
                - longueur focale (4 octets, float)
                - champ de vision (FOV, en degrés) (4 octets, float)

* bloc ATIP_CHUNK_CAMERA_ROTATION (paramètres extrinsèques concernant
                                   la rotation) :
                - ID = 0x0410
                - matrice 3x3 de rotation, les colonnes représent
                    les vecteurs de la base (9 * 4 octets, floats)
```

3 Implémentation et résultats

3.1 Architecture de l'implémentation

Lors de l'implémentation, le soucis de portabilité est intervenu. Pour cela, une bibliothèque de fonctions nommée *libatip* contient toutes les fonctions nécessaires pour traiter l'image originale, créer les données 3D et gérer le fichier exporté, en lecture et écriture. Cette bibliothèque est indépendante de la plateforme car elle n'effectue que des calculs et utilise des fonctions standard pour la gestion des fichiers. La biliothèque de chargement et d'enregistrement d'images utilisée est DevIL, elle est multi-plateforme. Grâce à tout ceci, le logiciel de conversion 2D vers 3D peut être porté sous Linux par exemple. Le navigateur peut aussi être porté sur de nombreuses plateformes, de bureau ou mobiles. Le fichier utilisé par le navigateur contient les textures dans le format désiré, la bibliothèque *libatip* peut ne pas être utilisée dans ce cas, facilitant l'accès aux plateformes mobiles.

Le développement de *libatip*, *ATIP maker* et *ATIP navigator* a été effectué sous *Visual Studio .NET 2003*. L'espace de travail avec cet environnement de développement est conçu avec une arborescence séparant les différents types de fichiers pour la clarté. Parmi ces répertoires se trouve la documentation générée par *Doxygen*, elle documente *libatip* de deux manières : pour les développeurs (documentation complète du code source), et pour les

utilisateurs de *libatip* (présentation des fonctions accessibles de l'extérieur). Tout le code source, les commentaires, la documentation et les deux logiciels sont écrits an anglais de manière à être facilement accessibles aux futurs développeurs.

3.2 Optimisations

Pour le traitement de l'image originale dans le but d'obtenir des données 3D, la vitesse d'exécution a été un critère déterminant lors du développement, certains algorithmes ont été choisis en se basant particulièrement sur ce critère. Nous ne détaillons pas les optimisations de bas niveau présentes sur toute la chaîne de traitements, mais nous citons celles qui ont un grand impact sur la vitesse d'exécution globale.

La principale optimisation qui a un très grand effet sur le temps de calcul global est le sous-échantillonnage de l'image originale. Elle consiste à utiliser une version à résolution réduite de l'image pour effectuer quasiment tous les calculs ; l'image à pleine résolution n'est utilisée que pour le calcul des textures de manière à en garder la précision dans le rendu 3D. Dans *ATIP maker*, la vitesse de calcul est déterminée par les options, 3 choix sont possibles :

- mode *normal* : l'image originale à pleine résolution est utilisée pour réaliser les calculs ; ce mode est utilisé pour les petites images (inférieur au mégapixel environ) et est très lent pour des images à haute résolution,
- mode *fast* : les dimensions de l'image originale sont divisées par 2, soit 4 fois moins de pixels ; ce réglage est adapté pour la plupart des photographies d'une appareil photo numérique,
- mode *very fast* : les dimensions de l'image originale sont divisées par 4, soit 16 fois moins de pixels ; ce réglage offre un temps de calcul faible, mais la précision des résultats est altérée ; à réserver pour les grandes images (à partir de 5 mégapixels environ).

Les résultats obtenus avec l'image de la figure 1 sont les suivants :

mode	temps d'analyse (secondes)	champ de vision calculé	erreur relative
normal	79	46.5 °	3.12%
fast	13	47.4 °	1.25%
very fast	3	46 °	4.17%

Ces temps d'analyse n'incluent pas le calcul des textures qui est indépendant du mode et qui dure environ 2 secondes pour une taille de texture maximum de 1024 pixels. Ces résultats sont obtenus sur un Athlon XP 1800+, 512Mo de RAM, avec une photographie en 2592×1944 (5 mégapixels). L'appareil photo numérique a précisé que le champ de vision est 48 °, ce qui a permis de calculer les erreurs relatives ; mais il faut prendre en compte la légère imprécision des données renvoyées par l'appareil qui a une influence sur le calcul d'erreur. En vision par ordinateur, étalonner la caméra est préférable par rapport aux données du constructeur. Nous pouvons nous apercevoir que le mode *very fast* comporte la plus grande erreur, résultat logique compte tenu du sous-échantillonnage important de l'image originale.

L'erreur apportée par le mode *normal* paraît étonnante, mais les différentes erreurs apportées lors des traitements peuvent avoir une influence sur le résultat. De plus, rappellons que l'appareil photo peut avoir renvoyé une valeur légèrement erronée et le résultat du mode *normal* est peut-être proche de la réalité. Enfin, le modèle de caméra que nous utilisons est approximatif, il ne tient pas compte des distortions radiales par exemple, ni du fait que le point principal n'est pas forcément au centre de l'image.

Il faut relativiser les erreurs obtenues : l'algorithme du *Tour Into the Picture* utilise une approximation importante de la scène (par un parallélépipède), et un étalonnage très précis de la caméra est inutile. Avec l'image de la figure 1, les différences entre les scènes 3D avec les trois modes de calcul sont indiscernables, car une légère variation de la focale ne modifie que légèrement les proportions de la scène. La matrice de rotation a une influence plus importante sur le résultat final, mais le sous-échantillonnage de l'image originale n'influe quasiment pas sur la qualité du calcul de cette matrice.

Le temps de traitement global est occupé en grande partie par la transformée de Hough permettant de déterminer les lignes de fuite (section 2.2.4). Le sous-échantillonnage de l'image originale a un très grand impact sur le temps d'exécution de cette transformée. Un ensemble d'optimisations de bas niveau ont été utilisées pour multiplier par 15 la vitesse d'exécution de cette transformée :

- *calculs en virgule fixe* : l'utilisation de cette représentation des nombres a un grand impact sur la vitesse. Le mode de représentation utilisé est : 16 bits pour la partie entière, 16 bits pour la partie décimale,
- *précalcul des tables de sinus et de cosinus* : les courbes à tracer ont pour équation $\rho = x\cos(\theta) + y\sin(\theta)$ sachant que θ est discrétisé de la même manière pour chaque courbe, le précalcul des sinus et des cosinus (en virgule fixe) est très utile pour la vitesse d'exécution,
- *rotation de la transformée* : les deux axes de la transformée sont inversés de manière à obtenir une ligne pour chaque θ, et un décalage en mémoire représente ρ ; les multiplications disparaissent avec cette méthode, permettant d'accélérer encore un peu les calculs.

Le facteur de 15 correspond au mode *normal* avec l'image de la figure 1 (dont la résolution est de 2592×1944 pixels). Le temps de calcul est d'environ 20 minutes sans les optimisations de bas niveau, d'où leur intérêt. Avec les deux types d'optimisation combinées, en mode *very fast*, nous obtenons donc un gain en vitesse d'exécution globale de $15 \times (79/3) \simeq 400$! Pour le mode *fast* (utilisé le plus souvent), le gain est d'environ $15 \times (14/3) = 70$.

3.3 ATIP maker

ATIP maker est la version pour Windows du logiciel pour convertir une photographie 2D en un monde 3D. Ce logiciel utilise la bibliothèque *libatip* pour effectuer ses calculs. Il est développé en C/C++, en utilisant les APIs Win32. Il utilise *OpenGL* pour le rendu de

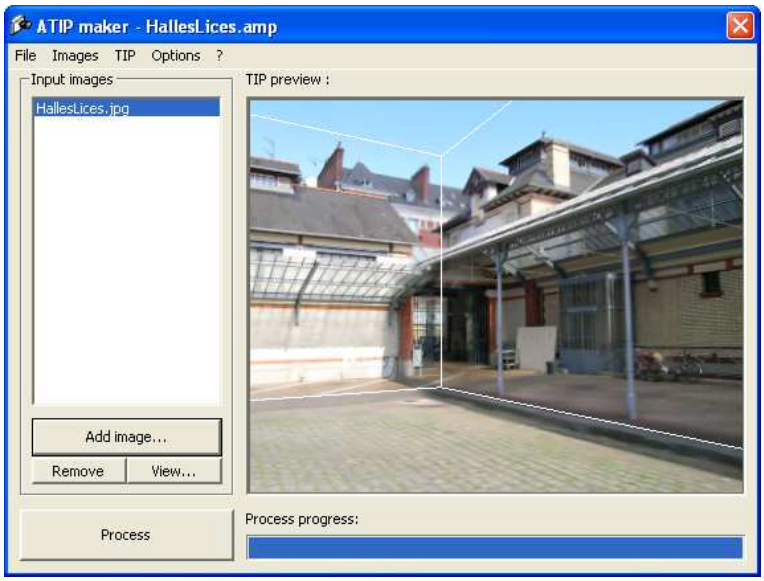

FIG. 31 – Fenêtre principale d'ATIP maker

la fenêtre de visualisation et pour l'éditeur de polygônes.

Sur la figure 31, nous pouvons voir la fenêtre principale d'*ATIP maker* après avoir calculé la scène. Dans la liste *Input Images*, le fichier *HallesLices.jpg* est l'image originale. La liste ne contient qu'un élément car le logiciel ne gère pour le moment qu'une seule photographie à la fois. La fenêtre de droite est la prévisualisation de la scène quand elle est disponible ; pour se déplacer et observer, il suffit de déplacer la souris en maintenant un des deux boutons appuyé, et enfoncer la touche *Ctrl* en même temps pour effectuer une translation latérale. Dans les menus, nous pouvons trouver toutes les fonctionnalités et options disponibles.

Sur la figure 32, nous pouvons voir la fenêtre d'options concernant la détection des points de fuite. Les différentes paramètres concernent principalement des seuils lors des différents traitements. Les paramètres par défaut fonctionnent pour une majorité d'images, mais des petits ajustements sont parfois nécessaires.

Dans la sous-fenêtre *Edge detection*, les options concernent la détection des contours (expliquées dans la section 2.2.3) :

- *Filtering method* est la méthode utilisée pour extraire les contours : *Zero-crossing pixels of the simple laplacian* est la méthode la plus simple, la plus rapide, celle qui marche pour la plupart des images. *Maxima detection of gradients* détecte les gradients de norme supérieure à un seuil ; cette méthode fonctionne bien mais avec une précision

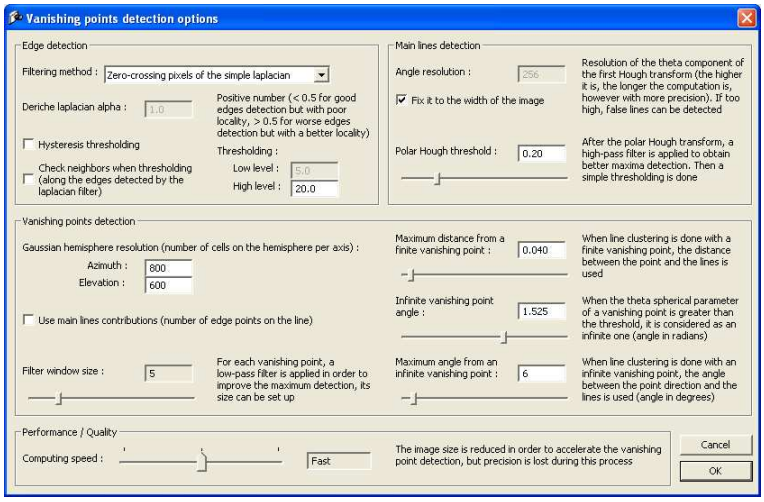

FIG. 32 – Panneau d'options pour la détection des points de fuite

inférieure à la précédente et un temps de calcul des traitements suivants supérieur à cause des contours épais obtenus. *Zero-crossing pixels of the Deriche laplacian* est une méthode expérimentale ne donnant pas des résultats convaincants, elle utilise un filtre récursif au lieu d'un filtre de convolution 2D classique (expliqué dans [17]).
- *Deriche laplacian alpha* est le paramètre permettant de modifier la qualité du résultat obtenu par le filtre laplacien de Deriche.
- *Hysteresis thresholding* permet d'utiliser un seuillage par hystérésis au lieu d'un seuillage simple (par défaut), cela permet de rajouter dans les contours finaux plus de petits contours qui peuvent avoir une influence intéressante pour la détection des lignes de fuite.
- *Check neighbors when thresholding* active l'utilisation du 8-voisinage des passages par zéros du laplacien pour effectuer le seuillage par hystérésis, sinon seuls les voisins qui sont aussi des zéros du laplacien sont utilisés (paramètre par défaut).
- *Thresholding Low level* : seuil bas du seuillage par hystérésis (en unités de norme de gradients).
- *Thresholding High level* : seuil haut pour les seuillages simple et par hystérésis (en unités de norme de gradients).

Dans la sous-fenêtre *Main lines detection*, les options concernent la détection des lignes de fuite (expliquées dans la section 2.2.4) :
- *Angle resolution* : résolution (angle θ) de la transformée de Hough polaire. Plus cette résolution est élevée, plus la précision de l'orientation des lignes de fuite est bonne.

Si la résolution est trop élevée, la qualité de la détection diminue fortement, et de mauvaises lignes sont détectées.
- *Fix it to the width of the image* fixe la résolution précédente à la largeur de l'image (réglage par défaut qui donne de bons résultats). Si l'image est sous-échantillonnée, la largeur de l'image réduite est utilisée.
- *Polar Hough threshold* : seuil appliqué après le filtre passe-haut sur la transformée de Hough. Ce seuil varie de 0 à 1 exclus, il influence beaucoup le nombre de lignes de fuite obtenu. Si la détection ne fonctionne pas sur une image donnée, ce paramètre est l'un des premiers à modifier.

La sous-fenêtre *Vanishing points detection* concerne la détection des points de fuite (expliquées dans la section 2.2.5) :
- *Gaussian hemisphere resolution* : résolution de la subdivision sur l'hémisphère gaussienne ; *Azimuth* est la résolution de φ, et *Elevation* celle de θ.
- *Use main lines contribution* : utilise les valeurs extraites de la première transformée de Hough pour assigner un coefficient aux lignes de fuite qui est le nombre de points alignés ayant permis de détecter chaque ligne. Ce paramètre a une faible influence et a une tendance à baisser la qualité de la détection des points de fuite ; certaines images peuvent en avoir besoin mais d'autres paramètres ont une influence supérieure.
- *Filter window size* : taille du noyau du filtre passe-bas appliqué sur l'image de la transformée de Hough sur l'hémisphère gaussienne pour chaque point de fuite ; 5 est la valeur par défaut et donne de bons résultats.
- *Maximum distance from a finite vanishing point* est la distance maximale en pixels entre une ligne de fuite et un point de fuite détecté pour considérer que cette ligne a contribué à la détection du point. Selon les images, il faut souvent augmenter ce paramètre par pas de 0.1 pour éviter d'exclure un trop grand nombre de lignes trop éloignées, mais qui ont tendance à fausser la position du point de fuite.
- *Infinite vanishing point angle* : angle θ sur l'hémisphère gaussienne à partir duquel un point de fuite est considéré comme infini (en radians). Il faut parfois baisser la valeur de ce paramètre pour éviter de considérer les points infinis comme des points finis, et obtenir un étalonnage de caméra moins précis.
- *Maximum angle from an infinite vanishing point* : angle maximal (en degrés) entre une ligne de fuite et une direction de point de fuite infini pour considérer que la ligne a contribué à la détection du point infini. Selon les images, il faut parfois augmenter ce paramètre pour éviter d'exclure un nombre de lignes trop élevé mais la précision de la direction du point de fuite baisse rapidement.

La dernière sous-fenêtre, *Performance/Quality* est abordée dans la section 3.2 et permet de choisir la vitesse de calcul : *Normal* utilise la pleine résolution de l'image originale, *Fast* divise les dimensions de l'image par 2 (soit 4 fois moins de pixels), et *Very fast* divise les dimensions de l'image par 4 (soit 16 fois moins de pixels).

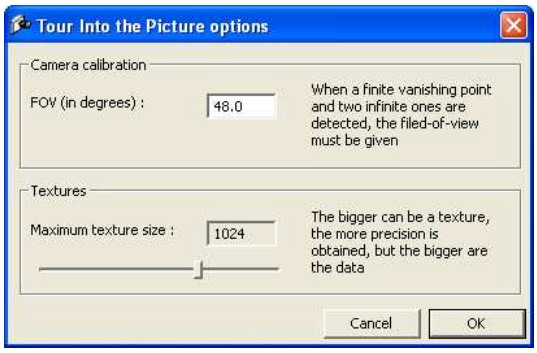

FIG. 33 – Panneau d'options pour le Tour Into the Picture

La figure 33 montre la fenêtre d'options pour le *Tour Into the Picture*. Cette fenêtre contient beaucoup moins d'options que la précédente car le traitement demande moins de paramétres manuels. L'étalonnage de la caméra a besoin d'un seul paramètre, et pour un seul des trois cas (décrit dans la section 2.3.1).

- *FOV* : champ de vision de la caméra (en degrés) pour le cas où un seul point de fuite fini est présent (et deux points de fuite infinis). La valeur par défaut est une moyenne du champ de vision obtenu couramment par les appareils photographiques. Une variation faible de ce paramètre n'a aucune influence visuelle sur le résultat final.
- *Maximum texture size* : taille maximale des textures en pixels. La taille des textures est calculée en fonction de l'aire sur l'image originale de la projection de la texture. La taille par défaut donne de bons résultats pour le navigateur en version PC. Pour un PDA, une résolution de 128, voire 256 donne de bons résultats pour un petit écran (simulable en redimensionnant la fenêtre du navigateur PC).

Voici les tailles de fichiers exportés obtenus en utilisant différentes tailles de textures pour la scène de la figure 1 (2465ko pour le fichier JPEG original) :

taille maximale des textures	taille du fichier exporté (en ko)
64	14.2
128	40.3
256	128.3
512	427
1024	1359
2048	3935

40.3ko sont nécessaires pour un fichier en qualité moyenne (taille maximale de 128 pixels) sur un PDA, ce qui est raisonnable en espace mémoire. Plusieurs scènes peuvent donc être stockées pour créer par exemple un systène de navigation.

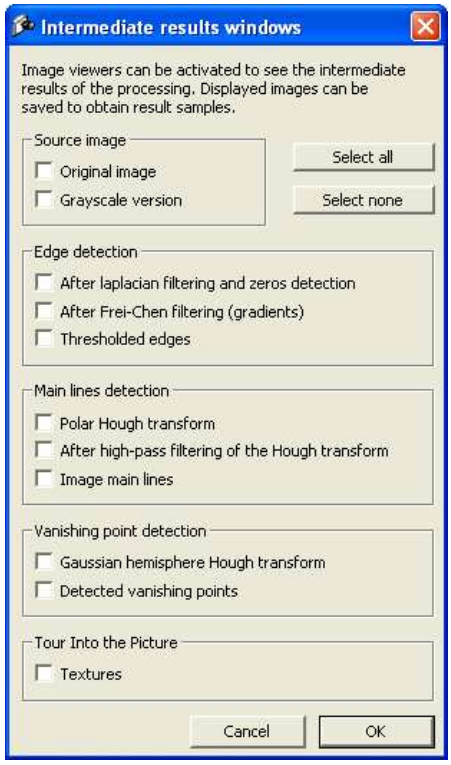

FIG. 34 – Panneau d'options pour l'affichage des résultats intermédiaires

La figure 34 présente la panneau d'options concernant l'affichage des résultats intermédiaires des traitements. L'activation de ces options permet d'afficher les images décrites ci-après :

- *Original image* : image originale, après sous-échantillonnage si les modes *Fast* ou *Very fast* sont sélectionnés.
- *Grayscale version* : version de l'image originale après la conversion en niveaux de gris.
- *After laplacian filtering and zeros detection* : image des passages par zéros pour les deux méthodes de détection de contours utilisant un filtre laplacien. Les points blancs représentent ces zéros.
- *After Frei-Chen filtering (gradients)* : images des gradients suivant les directions horizontales et verticales si cette méthode a été choisie pour la détection des contours.
- *Thresholded edges* : contours seuillés (par seuillage simple ou par hystérésis) représentés par des points blancs. Cette image est affichée pour les trois méthodes de détection des contours.
- *Polar Hough transform* : transformée de Hough permettant de détecter les lignes de fuite. Plus les pixels sont blancs, plus l'accumulation est forte. Les axes sont transposés pour une raison d'optimisation.
- *After high-pass filtering of the Hough transform* : transformée de Hough après filtrage passe-haut pour mettre en évidence les zones de forte amplitude et de faible rayon.
- *Image main lines* : image originale avec les lignes de fuite détectées en surimpression. Plus les lignes sont claires, plus le nombre de points alignés qui a permis de les détecter est élevé.
- *Gaussian hemisphere Hough transform* : transformée de Hough sur l'hémisphère gaussienne permettant de détecter les points de fuite. L'image obtenue est celle avant le filtrage passe-bas. Plus les points sont blancs, plus l'accumulation est forte. Les axes sont transposés pour une raison d'optimisation.
- *Detected vanishing points* : image originale avec les points de fuite et les lignes de fuite triées en surimpression. Les points de fuite finis sont représentés par des croix, les points de fuite infinis par une direction partant du centre de l'image. Les points et lignes rouges représentent le premier point de fuite, les verts le deuxième, les bleus le troisième, et les blancs ce qui est éliminé.
- *Textures* : textures calculées pour les différents polygônes de la scène (seulement ceux qui existent).

La barre de titre des différentes fenêtres contient un ensemble d'informations, comme par exemple le nombre de courbes tracées pour la première transformée de Hough. Chacune des fenêtres permet l'enregistrement de l'image par l'intermédiaire du menu, de manière à être exploitées.

Quand une image est chargée dans *ATIP maker* et que le traitement ne parvient pas à trouver les points de fuite, la modification de paramètres de détection des points de fuite est

nécessaire. Pour cela, l'affichage des résultats intermédiaires permet de savoir quels modifications doivent être réalisées. Voici une marche à suivre qui fonctionne bien en général :

- Afficher *Thresholded edges* pour observer le résultat de la détection des contours. Si des contours importants ne sont pas détectés, le seuil haut doit être baissé (*Thresholding High level*). A l'inverse, si trop de contours sont présents, avec beaucoup de bruit, le seuil doit être augmenté.
- Afficher *Image main lines*. Si le nombre de droites est trop important (recouvrant parfois l'image complète), le paramètre *Polar Hough threshold* doit être augmenté par pas de 0.1 dans un premier temps. Un nombre de droites entre 300 et 700 est en général correct.
- Afficher *Detected vanishing points*. Si des droites contribuant à des points de fuite finis ne sont pas détectés alors qu'ils apporteraient une contribution correcte, le paramètre *Maximum distance from a finite vanishing point* doit être augmenté par pas de 0.1 dans un premier temps. Si des points de fuite qui paraissent à l'infini sont considérés comme des points finis (indiqué dans la barre de titre un par la lettre F), le paramètre *Infinite vanishing point angle* doit être baissé. Si des droites contribuant à des points de fuite infinis ne sont pas classées alors qu'elles le devraient, le paramètre *Maximum angle from an infinite vanishing point* peut être augmenté, mais seulement par petit pas.

Le menu *TIP* d'*ATIP maker* contient quelques options de navigation et d'affichage :

- *Reset camera* permet de replacer la caméra à sa position d'origine. Si le mode *Align camera to the original camera* est sélectionné, la vue de la photographie originale est retrouvée.
- *Display scene frame* affiche en surimpression la scène en mode fil de fer pour comprendre sa structure.
- *Invert mouse Y axis* change le sens du mouvement pour l'axe vertical de la souris.
- *Align camera to the original camera* aligne la caméra virtuelle sur la caméra qui a permis de prendre la photographie. Un déplacement en avant suit la direction de vision du photographe. La scène n'est pas parallèle au monde 3D car le photographe n'est jamais parfaitement aligné avec le monde réel, surtout si un trépied n'est pas utilisé.
- *Align camera to the scene* aligne la caméra virtuelle avec les axes du monde 3D. De cette manière, un déplacement est réalisé suivant les axes, et permet l'observation sur les côtés sans effet de rotation indésirable.

ATIP maker contient un système de projet, nommé *ATIP Maker Project* (*AMP*) qui permet de retrouver le logiciel dans l'état où il était à un moment donné. Cela permet de créer par exemple un fichier de projet par scène qui doit être convertie, avec les paramètres associés (détection de points de fuite, Tour Into the Picture, fenêtres de résultats intermédiaires, résultats des calculs précédents). En chargeant un fichier d'extension *.amp*, les paramètres sont chargés tout comme la photographie originale (qui n'est pas incluse dans le fichier

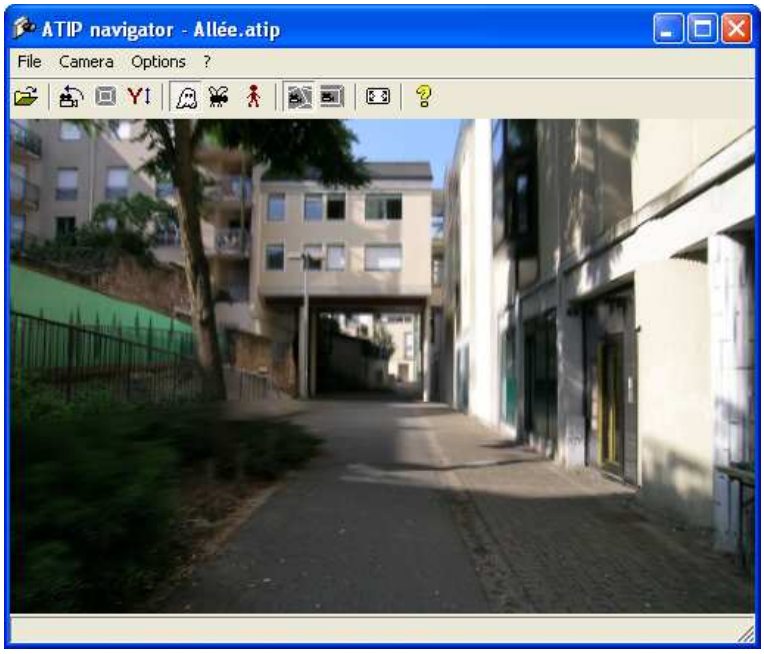

FIG. 35 – Fenêtre principale d'ATIP navigator

.amp), puis une boîte de dialogue demande si les textures doivent être calculées pour la prévisualisation.

3.4 ATIP navigator

ATIP navigator est la version pour Windows du navigateur 3D. Ce logiciel utilise la bibliothèque *libatip* pour gérer le format de fichier *.atip*. Il a été développé en C/C++ en utilisant les APIs Win32. Il utilise *OpenGL* pour le rendu 3D en mode fenêtré et en plein écran.

La figure 35 montre la fenêtre principale de *ATIP navigator*. Le déplacement à la souris dans la scène 3D fonctionne comme dans *ATIP maker*. Pour manipuler les différentes fonctionnalités, 3 possibilités sont offertes :

- le menu,
- la barre d'outils (avec des icônes),
- les raccourcis clavier.

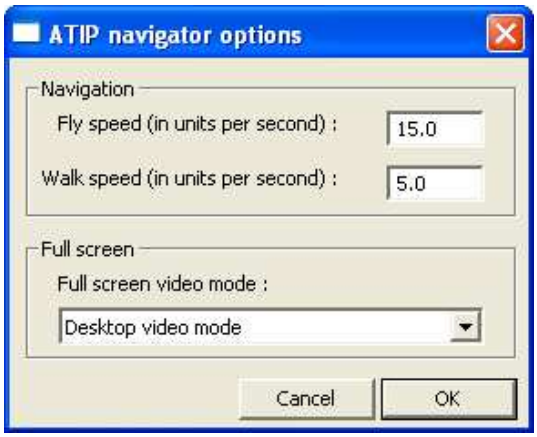

FIG. 36 – Panneau d'options d'ATIP navigator

Les fonctionnalités de la caméra (réinitialisation, alignement avec la scène, etc. ...) sont les mêmes que pour la prévisualisation de *ATIP maker*. Trois modes de navigation sont proposés :

- *Ghost mode* : mode par défaut, la navigation est libre dans toutes les directions,
- *Fly mode* : les mouvements de la souris permettent d'observer, et le clavier permet d'avancer,
- *Walk mode* : les mouvements de la souris permettent encore d'observer, mais les mouvements sont plus lents et sont bloqués à la hauteur du sol.

La figure 36 montre le panneau d'options d'*ATIP navigator*. On y trouve 3 paramètres :

- *Fly speed* : vitesse de déplacement en mode fly, en unités par seconde,
- *Walk speed* : vitesse de déplacement en mode walk, en unités par seconde,
- *Full screen video mode* : résolution, profondeur de couleurs, et taux de rafraîchissement du mode plein écran (change selon le matériel, avec détection automatique). La possibilité par défaut correspondant aux réglages courants du bureau.

L'option *Display a cube* permet de montrer l'utilisation possible de la réalité augmentée. Un cube transparent est placé sur le sol, et permet de bien voir le reste de la scène au travers.

4 Travaux futurs

Les travaux effectués représentent le point de départ d'un ensemble de travaux supplémentaires permettant d'améliorer les fonctionnalités existantes, et d'en rajouter des nouvelles. Quelques idées sont décrites ci-après, mais bien d'autres possibilités existent.

4.1 Images multi-résolution

Le format de fichier géré par le navigateur supporte n'importe quel format d'image, à condition d'implémenter son chargement. L'implémentation du navigateur peut se faire par exemple comme plug-in pour un navigateur Internet. Si les données transférées sont trop importantes, un chargement progressif des textures peut être envisagé pour obtenir un résultat très rapidement qui s'améliore ensuite étape par étape. Pour cela, il existe des formats de compression avec pertes qui permettent la gestion multi-résolution, comme *JPEG2000*. Cette possibilité est abordée dans [3].

4.2 Espace métrique

Avec l'algorithme existant, une unité correspond à la demi-hauteur de l'écran ; le monde 3D représenté n'utilise donc pas d'unité métrique. De plus, d'une image à l'autre, les dimensions obtenues sont différentes, génant la gestion des scène multiples (abordée plus loin). Si un facteur intervient, permettant de redimensionner la scène, une homogénéisation peut être obtenue. A partir d'une unique image, il n'est pas possible de déterminer ce facteur pour des raisons d'interprétation de l'image. La photographie peut représenter un petit objet tout comme une vaste scène.

Une solution est l'intervention de l'utilisateur : avec une fenêtre dédiée présentant la scène, il est possible de donner deux points, et la distance les séparant ; à partir de cela, les dimensions de la scène peuvent être déterminées. L'avantage principale de cette méthode est la possibilité de mesurer les objets. A partir d'une seule mesure, n'importe quelle autre mesure peut être déterminée, car les dimensions de la scène sont connues.

Cette méthode comporte des inconvénients, le principal est l'imprécision. Si par exemple les deux points fournis par l'utilisateur sont trop proches de la caméra, les mesures des objets en arrière-plan sont très imprécises. De plus, le modèle utilisé pour la caméra virtuelle ne tient pas complètement compte de toutes les distortions de l'appareil photographique. Enfin, si les mesures fournies ou demandées concernent des objets de la scène qui ne sont pas réellement dans les plans des différents quadrilatères, les mesures sont faussées. Les quadrilatères constituant la scène ne sont que des approximations : beaucoup d'objets de la scène (personnes, panneaux, voitures, ...) ne sont pas dans ces plans et sont donc déformés.

Le passage par un espace normalisé (métrique par exemple) n'a donc pas beaucoup d'utilité pour les mesures, mais se révélera important pour la gestion des scène multiples.

4.3 Gestion des scènes multiples

A partir d'une image, une scène peut être partiellement recréée. Cependant les possibilités de déplacements sont limitées, le réalisme décroît dès que la caméra virtuelle observe une zone non calculée. Pour cela, l'introduction de plusieurs scènes connectées les unes aux

autres permet d'augmenter le réalisme. L'usage qui peut en être fait est principalement un système de navigation : plusieurs photographies de la même scène, prises avec des orientations différentes, permettent, suite à des traitements adaptés, de reconstituer une scène de taille supérieure avec des possibilités de mouvements accrues. L'avantage de l'étalonnage automatique de caméra est la facilité des prises de vues : pas besoin de trépied, ni de mesures d'angle, toutes les photos peuvent être prises « à la main ». Le seul problème qui se pose est la normalisation de l'espace, de manière à pouvoir connecter les sous-scènes ensemble sans problèmes de dimensions.

Un autre usage peut être fait des scènes multiples : il s'agit de l'approximation d'une scène virtuelle (créée par un logiciel de modélisation par exemple). Si les rendus de la scène peuvent être effectués depuis des points précis, l'algorithme du *Tour Into the Picture* peut demander au moteur de rendu de créer les images nécessaires, puis les utiliserait pour créer un assemblage des différentes scènes. Les paramètres de la caméra étant connus car complètement contrôlés lors du rendu, les premières étapes d'*ATIP maker* peuvent être ignorées et l'éditeur de polygônes devient la première étape ; la projection de la scène dans des textures est l'étape cruciale de ces traitements. L'éditeur de polygônes pourrait comporter des traitements supplémentaires permettant de réduire l'intervention de l'utilisateur, car la topologie de la scène est complètement connue ; le cas idéal serait l'automatisation complète. L'avantage de cette méthode est la légèreté des données obtenues. Si une scène complexe comporte un nombre très élevé de polygônes, avec une grande quantité de textures, cette méthode permet d'obtenir une approximation intéressante, consommant peu de mémoire et peu de ressources processeur lors du rendu ; cette méthode est donc très intéressante pour les ordinateurs de poche (PDA).

La description d'un système à scènes multiples possible se trouve dans [5]. La transition d'une scène à l'autre se fait dès que l'observateur sort des limites d'une image. Il y est expliqué comment gérer le chargement progressif des scènes multiples par file d'attente, pour appliquer ceci à un plug-in d'un navigateur Internet.

4.4 Rendu de l'environnement

Les polygônes utilisés par *Automatic Tour Into the Picture* représentent l'arrière-plan de la scène ; les objets en avant-plan ne sont pas bien gérés, ils sont déformés. Cet usage peut-être appliqué au rendu d'environnements (*skybox*). Une scène 3D est rendue de manière classique (polygônes, textures, ...) et l'arrière-plan utilise une scène créée par *ATIP maker*.

A l'inverse d'une boîte d'environnement classique qui ne subit que des rotations simulant une image à l'infini, les translations des quadrilatères recréés, mêmes légères, donnent un effet de relief intéressant. Par exemple, à travers une fenêtre, la scène peut être observée, pas entièrement dues aux limitations de déplacements, et le léger effet de translation donne l'impression d'observer une scène proche et bien modélisée malgré la présence d'un nombre très faible de quadrilatères.

4.5 Objets en avant-plan

La méthode originale du *Tour Into the Picture* utilise des *billboards* pour afficher des objets en avant-plan. La méthode est présentée dans [1] et dans la section 1.2. La sélection des objets en avant-plan est réalisée manuellement avec un logiciel de traitement d'images. Pour *Automatic Tour Into the Picture*, il serait intéressant d'intégrer cette méthode de manière à augmenter l'effet de profondeur de la scène obtenue. Il est possible de conserver la méthode manuelle mais une automatisation du processus serait intéressante. Des travaux de recherche doivent être réalisés pour trouver le meilleur algorithme possible permettant de séparer les objets en avant-plan de ceux en arrière-plan dans l'image originale.

Nous remarquons que lors des traitements réalisés par *ATIP maker*, les objets en avant-plan sont « écrasés »sur le sol ou sur les murs, en général sans être alignés avec les axes du monde 3D. Les gradients des textures générées peuvent être exploités de manière à détecter les zones qui sont potentiellement des objets. Ensuite un algorithme de segmentation peut être utilisé de manière à détourer les objets. Une fois les objets extraits, l'arrière-plan comporte des données manquantes ; des algorithmes de remplissage sont en cours de développement et utilisent en général la cohérence spatiale pour prélever les données manquantes dans le voisinage des zones à compléter.

5 Conclusion

Automatic Tour Into the Picture marque le début d'un nouvel ensemble de travaux dans le domaine du rendu basé image. Ses avantages en terme de ressources (consommation mémoire, utilisation du processeur pour le rendu) le rendent très intéressant pour les plateformes mobiles en particulier. Les possibilités du *Tour Into the Picture* ont déjà été étudiées, mais l'automatisation du processus permet de simplifier l'utilisation de la méthode, tout en retirant des contraites très génantes de prise de vue. De plus, le temps de traitement global est réduit. Les résultats obtenus sont encourageants pour une approximation aussi importante de la scène photographiée. De nombreuses utilisations sont possibles : navigation, simplification du rendu dans des scènes 3D complexes, ... De nombreux travaux sont encore possibles pour augmenter le réalisme.

FIG. 37 – Exemple de résultats obtenus avec une allée de Rennes (la première image est la photographie originale)

FIG. 38 – Exemple de résultats obtenus avec un couloir (la première image est la photographie originale, dans la quatrième un cube de synthèse est ajouté)

Références

[1] Y. Horry, K.-I. Anjyo, and K. Arai. Tour Into the Picture : using a spidery mesh interface to make animation from a single image. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 225–232, 1997.

[2] K. Anjyo. "Tour Into the Picture" as a non-photorealistic animation. In *Computer Graphics*, volume 33-1, pages 54–55, 1999.

[3] N. Li and Z. Huang. Tour Into the Picture revisited. In *WSCG 2001 Conference Proceedings*, 2001.

[4] Siu-Hang Chu. Animating chinese landscape paintings and panoramas. Master's thesis, Hong Kong University of Science and Technology, August 2001.

[5] S. Yoon, H.-J. Chen, T. Hsu, and I. Yoon. Web-based virtual tour using the Tour Into the Picture (TIP) technique.

[6] H. Baltzakis and P.E. Trahanias. The VPLF method for vanishing point computation. *Image and Vision Computing Journal*, 19(6) :393–400, April 2001.

[7] V. Cantoni, L. Lombardi, M. Porta, and N. Sicard. Vanishing point detection : representation analysis and new approaches. In *CIAP01*, pages 90–94, 2001.

[8] N. Rasamimanana and G. Korngut. Effets du seuillage à hystérésis sur la détection de contours, 2001. ENST, http ://www.tsi.enst.fr/tsi/enseignement/ressources/mti/hysteresis.

[9] J. Lambourg and I. Ouachani. Influence de la taille des cases dans la transformée de Hough. http ://www.tsi.enst.fr/tsi/enseignement/ressources/mti/Hough.

[10] S. C. Lee, S. K. Jung, and R. Nevatia. Automatic integration of facade textures into 3D building models with a projective geometry based line clustering.

[11] C. Rother. A new approach for vanishing point detection in architectural environments, 2000.

[12] C. Bräuer-Burchardt and K. Voss. Robust vanishing point determination in noisy images. In *International Conference on Pattern Recognition (ICPR'00)*, volume 1, page 1559, September 2000.

[13] E. Lutton, H. Maitre, and J. Lopez-Krahe. Contribution to the determination of vanishing points using Hough transform. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, volume 16-4, pages 430–438, April 1994.

[14] E. Guillou, D. Meneveaux, E. Maisel, and K. Bouatouch. Using vanishing points for camera calibration and coarse 3D reconstruction from a single image. In *The Visual Computer*, volume 16-7, pages 396–410, 2000.

[15] S. C. Lee, S. K. Jung, and R. Nevatia. Integrating ground and aerial views for urban site modeling. In *16th International Conference on Pattern Recognition (ICPR'02)*, 2002.

[16] R. Cipolla, T. Drummond, and D. Robertson. Camera calibration from vanishing points in images of architectural scenes. In *BMVC99*, 1999.

[17] L. Guilbaud and P. Lacour. Déplacement des contours par passage par zéro du laplacien à l'aide de divers paramètres. http ://www.tsi.enst.fr/tsi/enseignement/ressources/mti/zero_du_lap/mti.html.

Unité de recherche INRIA Rennes
IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)

Unité de recherche INRIA Futurs : Parc Club Orsay Université - ZAC des Vignes
4, rue Jacques Monod - 91893 ORSAY Cedex (France)
Unité de recherche INRIA Lorraine : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)
Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)
Unité de recherche INRIA Sophia Antipolis : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399